conference
proceedings

LISA '09: 23rd Large Installation System Administration Conference

Baltimore, MD, USA
November 1–6, 2009

Sponsored by
USENIX and SAGE
in cooperation with
LOPSA & SNIA

USENIX
Proceedings of LISA '09: 23rd Large Installation System Administration Conference
Baltimore, MD, USA November 1–6, 2009

ISBN 978-1-931971-71-3

USENIX Association

Proceedings of LISA '09: 23rd Large Installation System Administration Conference

November 1–6, 2009
Baltimore, MD, USA

Conference Organizers

Program Chair
Adam Moskowitz

Program Committee
Paul Anderson, *University of Edinburgh*
Paul Armstrong
Travis Campbell, *AMD*
Narayan Desai, *Argonne National Laboratory*
Ian Dotson, *University of Washington*
Andrew Hume, *AT&T Labs—Research*
Brent Hoon Kang, *The University of North Carolina at Charlotte*
William LeFebvre, *Digital Valence, LLC*
Chris McEniry, *Sony Computer Entertainment America*
Mario Obejas, *Raytheon*
David Plonka, *University of Wisconsin*
Mark D. Roth, *Google, Inc.*
John Sellens, *SYONEX*
Gautam Singaraju, *The University of North Carolina at Charlotte*
Nicole Velasquez, *University of Arizona*

Invited Talks Coordinators
Doug Hughes, *D.E. Shaw Research, LLC*
Amy Rich, *Tufts University*

Workshops Coordinator
Lee Damon, *University of Washington*

Guru Is In Coordinator
John "Rowan" Littell, *California College of the Arts*

Posters Coordinator
Gautam Singaraju, *The University of North Carolina at Charlotte*

Steeering Committee
Paul Anderson, *University of Edinburgh*
David N. Blank-Edelman, *Northeastern University*
Mark Burgess, *Oslo University College*
Alva Couch, *Tufts University*
Æleen Frisch, *Exponential Consulting*
Xev Gittler, *Morgan Stanley*
William LeFebvre, *Digital Valence, LLC*
Mario Obejas, *Raytheon*
Ellie Young, *USENIX Association*
Elizabeth Zwicky, *Consultant*

USENIX Board Liaison
Alva L. Couch, *Tufts University*

The USENIX Association Staff

External Reviewers

Bob Apthorpe
Kenytt Avery
Lori Barfield
Matthew Barr
Lois Bennett
Bill Cheswick
Marc Chiarini
Alva Couch
Jason Faulkner
Esther Filderman
Jon Finke
Scott Francis
David Harnick-Shapiro
Peter Jansson
Gabriel Krabbe
Tom Limoncelli
Cat Okita
Dustin Puryear
Matthew Sacks
Josh Simon
Chris St. Pierre
Marc Staveley
John Stoffel
Leon Towns-von Stauber
Rudi van Drunen
Elizabeth Zwicky

LISA '09: 23rd Large Installation System Administration Conference
November 1–6, 2009
Baltimore, MD, USA

Message from the Program Chair v

Wednesday, November 4

The Human Side of Sysadmin

Pushing Boulders Uphill: The Difficulty of Network Intrusion Recovery 1
Michael E. Locasto, George Mason University; Matthew Burnside, Columbia University; Darrell Bethea, University of North Carolina at Chapel Hill

Two-Person Control Administation: Preventing Administation Faults through Duplication 15
Shaya Potter, Steven M. Bellovin, and Jason Nieh, Columbia University

The Water Fountain vs. the Fire Hose: An Examination and Comparison of Two Large Enterprise Mail Service Migrations 29
Craig Stacey, Max Trefonides, Tim Kendall, and Brian Finley, Argonne National Laboratory

Thursday, November 5

Networks, Networks, Networks

Crossbow Virtual Wire: Network in a Box 47
Sunay Tripathi, Nicolas Droux, Kais Belgaied, and Shrikrishna Khare, Sun Microsystems, Inc.

EVA: A Framework for Network Analysis and Risk Assessment 65
Melissa Danforth, California State University, Bakersfield

An Analysis of Network Configuration Artifacts 79
David Plonka and Andres Jaan Tack, University of Wisconsin—Madison

Security, Security, Security

Secure Passwords Through Enhanced Hashing 93
Benjamin Strahs, Chuan Yue, and Haining Wang, The College of William and Mary

SEEdit: SELinux Security Policy Configuration System with Higher Level Language 107
Yuichi Nakamura and Yoshiki Sameshima, Hitachi Software Engineering Co., Ltd.; Toshihiro Tabata, Okayama University

An SSH-based Toolkit for User-based Network Services 119
Joyita Sikder, University of Illinois at Chicago; Manigandan Radhakrishnan, VMware; Jon A. Solworth, University of Illinois at Chicago

Friday, November 6

On the Fringe

Federated Access Control and Workflow Enforcement in Systems Configuration 129
Bart Vanbrabant, Thomas Delaet, and Wouter Joosen, K.U. Leuven, Belgium

CIMDIFF: Advanced Difference Tracking Tool for CIM Compliant Devices 145
Ramani Routray, IBM Almaden Research Center; Shripad Nadgowda, IBM India Systems and Technology Lab.

Transparent Mobile Storage Protection in Trusted Virtual Domains 159
Luigi Catuogno and Hans Löhr, Ruhr-University Bochum, Germany; Mark Manulis, Technische Universität Darmstadt, Germany; Ahmad-Reza Sadeghi and Marcel Winandy, Ruhr-University Bochum, Germany

Message from the Program Chair

Dear LISA Attendee:

Back in 1989, LISA was just a small, two-day workshop, while the conference we now call the USENIX Annual Technical Conference was held twice a year and was known as "Summer USENIX" or "Winter USENIX." Most of the attendees were programmers, often researchers, but some of us were either dual-role programmer-sysadmins or dedicated sysadmins. The papers were mostly about operating system research, but a few were clearly of interest to system administrators: for example, Brent Callaghan had just introduced the automounter, two guys from BBN presented an implementation of dial-up IP for UNIX, people from NYSERnet had an implementation of SNMP, and Denise Ondishko from the University of Rochester delivered her paper "Administration of Department Machines by a Central Group." In 1990, LISA changed from a "workshop" to a "conference," but many LISA attendees remained regulars at the general conference. LISA added a third conference day in 1991. By the time I went to my first LISA, in 1994, a handful of tutorials had been added, on the Monday and Tuesday before the conference.

We've all come a long way since 1989. LISA is now one of USENIX's flagship conferences, with six days of tutorials, three days of workshops, and three days of multitrack technical sessions. LISA papers have moved from talking about line printer systems and early implementations of what would eventually become sudo, to automated virtual networks, high-level languages to make dealing with SELinux easier, and protection schemes for USB keys using trusted virtual domains. Most importantly, most of the LISA attendees are dedicated, professional system administrators rather than programmers who drew the short straw and got "taking care of the systems" added to their regular duties.

Some things about LISA don't change. In particular, it takes a small army of people to make the conference happen—far too many to name or count here. Let's just say that a quick count puts the number well over 200, and that doesn't include the on-site staff. This conference wouldn't be possible without all these people. Compared to all of them, my role as program chair is more of a figurehead and high-level organizer than anything else. As you move about the conference, please take a moment and say "thanks" to anyone you see wearing an official badge holder or ribbon.

Of the 38 papers submitted (or 34, depending on how you want to count them), we accepted 12. Those papers, the full text of which appears in these proceedings, are a relatively small but significant part of the conference, and, like those of every LISA past, represent the best research and "deep thought" about system administration today. I encourage you to read every one of them. I also encourage you to make an effort to meet new people, to listen to a talk or paper that covers material outside your usual interests, and to attend a LISA activity that's new to you. Finally, remember to have fun. After all, how many times do you get to hang out with hundreds of people who all understand what you do for a living, who know the same acronyms you do, and who don't think you're weird? Take advantage of this opportunity while you can!

Adam Moskowitz
Program Chair

Pushing Boulders Uphill: The Difficulty of Network Intrusion Recovery

Michael E. Locasto
George Mason University
mlocasto@gmu.edu

Matthew Burnside
Columbia University
mb@cs.columbia.edu

Darrell Bethea
UNC Chapel Hill
djb@cs.unc.edu

Abstract

One of the most significant unsolved problems for network managers and system administrators is how to repair a network infrastructure after discovering evidence of an extensive compromise. The technical issues are compounded by a breathtaking variety of human factors. We present a study of three significant compromises of a medium-scale network infrastructure. We do so as a way to expose the difficulties — both technical and human — inherent in intrusion recovery. Most network users take a "secure" network infrastructure for granted. Real events show that this level of faith is unwarranted, as is the belief that intrusions are or can be completely repaired, especially in the absence of research on network recovery mechanisms that explicitly take the needs of support staff into account. We conclude with lessons learned and some detailed suggestions for tools that can help bridge this gap.

"Damage control is much easier when the actual damage is known. If a system administrator doesn't have a log, he or she should reload his compromised system from the release tapes or CD-ROM."
– Firewalls and Internet Security:
Repelling the Wily Hacker [6].

1 Introduction

This paper presents a case study of the impact of social pressure, technical experience, bias, and other constraints on both individual and group risk assessment and decision-making during the recovery efforts from three significant network intrusions at a single site in March of 2007, December of 2007, and March of 2008.

Although many people enjoy the benefits of access to information and communication through networked systems, most take the security and reliability of these infrastructures (residential ISPs, workplace IT departments, the IT infrastructure of educational institutions, *etc.*) for granted. Users do not often see the impact of computer break-ins and intrusions beyond the occasional sensational story that reaches the front page of some major news outlet. Skilled attackers work hard not to be noticed. System administrators worth their salt work even harder to make sure intrusions are prevented. Institutions have deep concerns about negative publicity.

As a result, users have a misguided understanding of the frequency of such attacks and the difficulty of maintaining and repairing a network. Users may incorrectly assume that IT staff can fully repair the damage or harm (think of copied intellectual property, computer cycles used, reputations lost) caused by an attacker. Even researchers in the systems security space may summarily dismiss the task as a simple, if somewhat lengthy, system administration job, and thus unworthy of investigation. It is our opinion that the problem of coordinating the repair and restoration of network infrastructures is a major unaddressed problem that embeds a number of unanswered research questions involving the intersection of human factors and technical challenges.

1.1 Dual Nature of the Problem

Compromises of medium or large networked systems (such as the infrastructure supporting a research department, college, or university) are difficult to analyze and respond to for a number of reasons. As a result of the diversity of the problem and the lack of research into methods that deal with both technical and human factors, network intrusion recovery is more of an art than a science. The state of the art often involves manually reinstalling machines from read-only media, as the traditional text on firewalls [6] reminds us in the quote above. Even when this process *is* automated, it still resets systems to some initial state, thus deleting valuable data that may not have been backed up, or information that would be of some

use in a forensic investigation. At this point, we must resist the temptation to treat the problem as solved by turning to some technical solution (*e.g.,* automated network-based OS installations, "ghosting" software, or recent research on an automated process for working backward from the attack to undo the damage caused [13, 7]). Both technical and human factors introduce obstacles that simply executing a software application cannot overcome.

Even with the assumption that we can reliably detect an intrusion, there are many technical issues related to repairing a wide variety of hosts, nodes, objects, and artifacts. These issues, and the decisions necessary to address them, are compounded by a number of human factors. The workflow we depict in Figure 1 and the issues listed below are representative rather than exhaustive.

First, even with deep auditing information, it can be difficult to describe the extent of an intrusion within the context of a single system. Second, determining the extent of the damage throughout the network requires replicating or extracting those conditions to widen the scope of the detection process. Once the process of detecting an attack and determining its scope have been accomplished, then the process of recovery presents an overwhelming series of choices and possibilities. As we can see from the incidents described later in the paper, this process is not strictly linear. Thinking of detection as "accomplished" rather than "ongoing" is misleading.

Planning and implementing a recovery can involve a variety of changes to systems, hardware, applications, and network topology. Individual systems require forensics and may need to be isolated, removed, updated, or reconfigured. Software applications may need to be reconfigured or have patches applied, which raises the twin issues of which applications to fix in what order and what patches to generate or obtain (and what order to apply them). The network topology may need to change: new routers, switches, or other equipment may need to be introduced or existing equipment reconfigured. Firewall rules may need to be introduced or modified. Existing IDS sensors could require retuning. During this entire process, the team must test and verify each step.

We begin to see recovery as a complicated, fluid process. Response teams often labor under a compressed time frame to fix as large a part of the intrusion in as short a time as possible. The forensics process experiences pressure to finish quickly to reduce service downtime. The recovery team's training and skill level, along with the vagaries of interpersonal relationships, can constrain what types of actions are realistic. Promotion, demotion, hiring, or termination decisions can affect someone's willingness to engage in extensive recovery actions. In addition, attacks rarely occur at convenient times; if the incident occurs near social events or holidays, time pressure can greatly increase.

Although some technical fixes may be "obvious", both internal (to the team) and external (*i.e.,* the team's customers and employers) vested interests in maintaining the network status quo can prevent the implementation of these fixes. The team must be familiar with the preferences, attitudes, and biases of the user or customer population in order to "sell" the repair to them. Finally, the reputations of the team, individuals, customers and users, and institution requires careful consideration.

1.2 Contributions

This paper offers evidence that illustrates what might otherwise be an overlooked point by information security researchers: intrusion recovery is not a simple systems administration task. Intrusion recovery, while a large technical challenge, is further complicated by human–level issues, and we highlight specific issues involved in the incidents we describe. In addition to our analysis, we provide the research community with three real (rather than artificial, contrived, or based on conjecture) threat and recovery scenarios. Intrusion recovery systems are relatively neglected in the research literature; we believe the community should focus on creating mechanisms that deal with recovery as a system composed of both humans and computer systems.

1.3 Background and Related Work

Complete technical solutions to the problem of recovering from *realistic* intrusions in the research literature are sparse, although both classic [25, 24, 5] and more recent [23, 11] examples of post-mortem intrusion analysis do exist. Spafford's analysis [24] of the Morris Worm and Cheswick's annotated log of the Berferd case [5] can be seen as catalysts for changing the way computer scientists and network researchers thought about trust and security on the fledgling Internet. The analysis of these incidents helped spur the adoption of stronger authentication mechanisms, the use of firewalls to implement host communication policies, and research on basic auditing tools and intrusion sensors. Singer [23] recounts how even a well-designed infrastructure managed by an experienced, professional network security team can be compromised. This latter analysis helps illustrate just how difficult and time-consuming it can be to completely remove an attacker from a system. In particular, the attacker described in Singer's article would repeatedly find another avenue into the infrastructure just when the admins thought they had adjusted their security posture appropriately.

The HotAdmin[1] project at UBC has looked at the nature of the job of security administrators [10]. They compare the dynamics of a centralized and distributed secu-

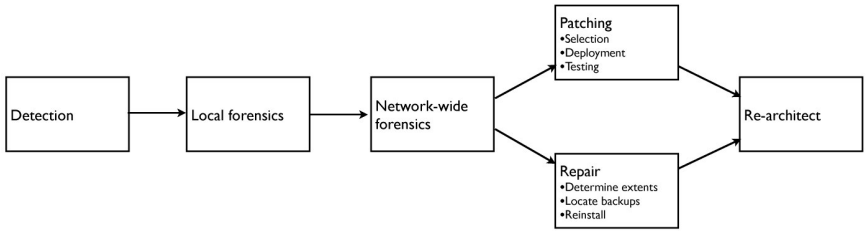


Figure 1: *Response flow.* Our attempt to define a workflow for a technical response at a high level of abstraction. Decision trees at individual parts of this workflow can be partially hidden or incomplete with very large branching factors. This figure belies the fluidity with which a response scenario can take place; detection, diagnosis, and reaction do not form a strictly rigid, step-by-step process. In searching for a way to more easily visualize the relationships between these activities, we compromised at a high level of abstraction.

rity group at an academic organization, and how the transition between the two models worked.

Much research takes a prophylactic stance: networks and systems should be hardened before an attack occurs. Needless to say, proactive hardening, even if it provides strong protection mechanisms like tainted dataflow analysis [16, 26] only partly addresses the problem: the cost of use may be high, the adoption rate low, and the "coverage" of the technique (in terms of classes of attacks defended against) narrow. To date, only memory address space randomization [2] seems to have seen significant deployment, but even this protection mechanism only addresses a certain class of attacks. Other efforts to deal with intrusion recovery discuss ways to provide secure backup, alert logging [7, 19], and audit systems [21]. Meanwhile, Kursawe and Katzenbeisser [14] argue that the prophylactic stance is limited. They introduce a new paradigm where computer users accomplish useful work even though their machines are compromised.

The problem itself appears too large for a single, comprehensive technical solution [9]. System administrators, therefore, are relegated to selecting a hodgepodge of sensors and countermeasures to help defend their networks and restore order when intrusions are finally noticed. The selection process is driven by a variety of possible considerations — not just purely technical issues. These considerations may range from cost and resource constraints on equipment and personnel to "political" factors, personal experience, or recommendations from friends or colleagues. These factors can exert a powerful influence. Although neither of the following cases apply in the incidents we describe, it is not difficult to imagine such situations. For example, a faculty member may have had a role in developing a particular networking technology or intrusion sensor, or an IT company feels obligated to use only their OS or toolset.

Defending a network involves assessing risk and allocating resources to match the perceived threats and costs [4]. In terms of network intrusion recovery, knowing that the network is at a high risk of a compromise does not directly inform the procedures that should be in place for repair. Instead, it may inform strategies for reducing or managing risk, and little research exists on systems for managing the disaster workflow recovery once a network *is* compromised.

The psychology community has spent a significant amount of time studying and trying to understand the process of human decision making under duress. Payne *et al.* [20] provide a good overview of the research in this area, including beliefs about uncertain events, decisions made under risk and uncertainty, and frameworks for decision behavior. Consideration of how security-related decisions are made under stress seem to fall most naturally into discussions about the threat model a system operates under, as bad decisions by the system user could increase the power of the hypothetical attacker.

Finally, as we saw in our attempts to collect information for this case study, the human memory and recollection is notoriously unreliable. The reliability of eyewitness [27] and earwitness [3] testimony has been extensively studied by psychologists; in fact, Wells and Olson [27] point out that the only scientific body of literature on eyewitness reliability exists in the psychology space. In the computer security field, and in the context of rebuilding complex network infrastructures and carrying out a number of both repetitive and complex tasks over a long period of time, human memory is relied upon far too much. Our case study shows that it is possible for initial planning goals, suggestions, or objections to be misunderstood, warped, or forgotten — leaving potentially large gaps in the actual level of security achieved after repairs complete.

2 Methodology

In researching this paper, we interviewed all parties involved in recoveries from three recent attacks on a medium-scale network, including administrative staff and management. We performed an email archive search, and confirmed many of the details of the attack through analysis of disk images from a number of the compromised machines. We performed an initial debrief of the entire IT staff and followed up more extensively with four of the IT staff members. We have continued monitoring the organization's response.

We emphasize that we do not aim to lay blame with individuals, and we refrain from naming the people and organization involved. Each interview subject gave us permission to interview them and report on the process. Our goal is to present the facts of the situation, disposition of the network, and decisions made by the staff in as clear a light as possible as a way to motivate research and development of tools that ease the burden on IT staff during the process of network intrusion recovery.

One of the most significant challenges when responding to an intrusion is performing forensic analysis to determine the exact impact of the attack. In the case of the compromises we discuss here, the nature of the attacks was such that no individual performed a single coherent analysis. Rather, the analysis was performed piecemeal by the various members of the IT staff, and, as such, each had a different view of the impact of the attack. As we discuss later, this fractured view presents the IT staff with problems when attempting to form a coherent response. Furthermore, it presents a problem to us as researchers. In some cases, parties we interviewed had radically different timelines and analysis, even though the interviews took place less than a month after the attacks of December 2007 and within the scope of the March 2008 attacks. Where possible, conflicting statements were reconciled through mechanical methods (email or file modification dates) but some ambiguity remains.

3 Intrusion Incidents

The network on which we focus our attention in this paper is the network for a mid-sized research department at a large university. The network consists of approximately 1000 Windows, Linux, and Solaris workstations, as well as a number of infrastructure servers providing DNS, DHCP, and HTTP, and several general purpose compute clusters accessible via SSH. Approximately 150 of the workstations run Red Hat Enterprise Linux (RHEL) AS v4. These machines are periodically updated from two source machines using `rdist`. For the purposes of load balancing, the two `rdist` masters are each responsible for half of the machine population. User authentication in this environment is centralized. Windows machines authenticate users via Active Directory; the Linux and Solaris machines authenticate users through NIS. At the time of these incidents, the network employed neither a rule–based IDS like Snort nor an anomaly sensor. As a partial result of these incidents, the network will shortly employ a content–based network anomaly sensor. Machines are generally not firewalled (although most end hosts have a local firewall supplied by their OS vendor). The network supports a research environment with a strong tradition of open access. This tradition supplies a political force that has precluded the use of any form of firewall at the network edge. One of our colleagues (not associated with these incidents) pointed out that a firewall is merely a device for implementing policy. If the policy is unclear, then the mere presence of such a device is unlikely to help.

Over the time period covered in this case study, the network was administered by an IT staff of three to five people, with a single manager. This staff works independently within the context of the larger IT organization of the university. The IT manager is highly experienced in managing staff and infrastructure and had previously completed a vast overhaul and update of the infrastructure to bring some amount of order to what was an otherwise disorganized physical and virtual space.

There is a high turnover, and staff members come from widely disparate backgrounds; some are students with little to no experience, while some are highly knowledgeable and very experienced. The network is complex for its size and has a number of systems, including the accounting system, which remain unchanged from the late 1990s. New staff, even if highly experienced, often take months to gain a complete understanding of the intricacies of the network.

3.1 March 2007 Attack

In March of 2007, an attacker attempted to use a kernel exploit to gain root privileges on several of the RHEL workstations. The attack was discovered when several of these attempts failed, raising alerts. For each machine on which the attack failed at least once, the IT team were able to use system logs to determine the origin of the attacker and the compromised user accounts he was using to access the machine.

The failed attacks were not all the same, however; the attacker was revising his methods, and there was no way to determine if he had succeeded. The staff checked the logs of other susceptible machines (those harboring the same vulnerability, but showing no indication of failed attacks). While staff could uncover no indication that the attacker had connected to the machines, it is possible that he altered the logs after gaining root access.

3.2 March 2007 Response

It is possible that the attacker never succeeded. Regardless, the safest response in this situation, recognized by all members of the IT staff, would have been to reinstall all vulnerable machines with a patched version of the operating system. There were, however, external constraints that prevented this approach. The attacks occurred in the middle of the semester and involved many machines heavily used by classes. Thus, the staff needed to carry out a solution as quickly as possible to avoid disruption to the Department's academic mission.

Most of the systems are nearly identical, with the exception of the servers and the `rdist` masters. Reinstalling the `rdist` masters would have been time consuming and error-prone, as the `rdist` distribution architecture in use was archaic and proprietary, and those most familiar with it were no longer employed.

Furthermore, reinstalling the workstations using the `rdist` new-install process would have taken far too much time, as each install generally took about a half day, and due to network bottlenecks (much of the install was network-based), no more than four or five machines could reasonably be installed at any given moment.

The IT staff's primary insight was that there were two classes of vulnerable machines: servers and workstations. The attack required a user-level shell account on the target machine in order to work, and the attacker had compromised at least one or two student accounts (as indicated from the logs of the failed attacks). Student accounts, however, do not have access to the servers, so the likelihood of an infection on those machines was less than on any given workstation, as long as the staff assumed that the attacker had not compromised any administrator accounts. The workstations, on the other hand, were mostly identical, only differing in a few configuration files. By isolating those files, the staff believed they could clone workstations from other workstations and avoid the bottleneck to the master `rdist` servers.

The staff shutdown each server and ran several rootkit checkers. They also performed some manual log inspection for any indication of an attack. Seeing none, they patched the servers and brought them back online.

The staff then performed a standard (half-day) new install on a single workstation via the master server. While this new, clean workstation was installing, the staff used the time to analyze workstations of many different configurations to determine the minimal set of configuration files that would differ per machine. They also burned approximately twenty Linux LiveCDs. Once the first workstation was finished installing, the team went to each remaining workstation, booted to a LiveCD, and inspected the configuration files which were to be left untouched to verify that they contained nothing malicious.

The staff members then downloaded and ran a script from the local intranet. This script erased most world-writable locations on the machine (`/tmp`, parts of `/var`, etc.). It then synchronized the remainder of the local filesystem (with the exception of the wiped partitions and the workstation-unique configuration files) directly from a known-clean workstation. Staff then re-configured and re-installed the bootloader and restarted the workstations.

Once the single clean workstation had been cloned, it was possible to use the newly cloned machines themselves as `rdist` masters for other machines. For example, by choosing masters within the same room, on the same local switch, it allowed for a dramatic decrease in the amount of time for the entire recovery. Note the level of detail and manual effort involved in starting and evolving the repair and recovery process, including a heuristic learned only through direct experience with reinstalling machines in a localized fashion.

At this time, staff considered the problem resolved and returned to normal day-to-day operations. However, we saw in our interviews that some members of the staff recognized, even at the time, that they were unsure whether the attack had been truly cleaned up. Furthermore, there was no record keeping and no analysis or formal discussions regarding installation of additional security measures such as an intrusion detection system.

3.3 December 2007 Attack

Early in 2007, four new machines arrived at the department, intended for use in high-performance graphics research. Each machine was equipped with a high-end NVIDIA graphics card. No official Linux drivers for these graphics cards existed, so staff used unofficial drivers. In early December 2007, all four machines stopped working. The IT staff installed updated (and now official) graphics drivers, which solved the problem until all four machines crashed the next day.

The staff pushed out updates to all RHEL machines through its two `rdist` servers, `starsky` and `hutch`. `starksy` is the primary master `rdist` server and `hutch` is a secondary. The infrastructure accomplishes upgrades with a two stage process. In the first stage, the active RHEL installation on `starsky` is upgraded. This live operating system is manually imaged and the image copied to `hutch`. A `cron` job on each of these machines pushes the upgraded image out to half of the 150 machines. The unfortunate consequence of this architecture is that a compromise on `starsky` would be pushed out automatically to the entire network. The staff installed the updated NVIDIA driver on `starsky` to prevent it from being overwritten on the graphics machines after the next `rdist`.

In addition to handling the NVIDIA issue, the staff also upgraded the kernel on `starsky` from version `2.6.9-55.0.9.EL` to `2.6.9-55.0.12.EL`. At 4 AM, the cron job delivered the upgrade to all 150 machines. On 10 December 2007, the staff discovered that both `starsky` and `hutch` had crashed. The staff attributed the failure to the recent upgrade, and investigating it was added to the end of a long task list for one of the staff members. Both machines crashed again on several subsequent nights.

The issue was finally explored on 13 December 2007, and the recent patches were rolled back on `starsky`. That night, both machines crashed again. This was a strong indication that the patches were not the problem, so an attempt was made to re-upgrade `starsky`. The upgrade failed when, during kernel compilation, the `mkdir` command returned an error. On the morning of 17 December 2007, exploration of this error determined that `mkdir` failed when attempting to create directories consisting only of numeric characters. IT staff began to suspect a rootkit. Booting to a LiveCD confirmed that suspicion: several files, including `mount`, had been replaced.

The hypothesis of the IT staff is that the rootkit installed by the attacker conflicted with the kernel module of the NVIDIA driver. If the attack took place in the first week of December, the rootkit would have been pushed to the graphics machines, a conflict ensued, and the machines crashed. Installing the driver on `starsky` caused that machine to crash too. The near-simulaneous kernel update obscured the real issue.

3.4 December 2007 Response

Discussion and planning for the response took place in a hallway at around 1pm on 17 December 2007. The planning group was assembled informally and consisted of the IT manager, three IT staff, and two authors of this paper, who happened to be nearby.

Initial discussion surrounded disagreements on the scale of the attack and the nature of the exposure. There was a brief argument over whether the `rdist` servers could be re-imaged and a clean install pushed out to all machines. This idea was discarded because it was recognized that all 150 machines would have to be reformatted from scratch. Planning began on how that process would take place, and a number of questions were raised immediately. What, if any, changes should be made to the system architecture? If changes are made, in what order, and to which machines, should those changes be rolled out? Who will be involved? Staffing shortages imply that any changes beyond the simplest would take weeks or months to put in place. How will changes affect end users? Finals week is in progress, so taking large numbers of machines offline is undesirable.

Discussion immediately centered around whether the staff should either stick with Red Hat Enterprise Linux or move the machines to another operating system. We note that were was no *a priori* reason to blame RHEL for the intrusion, and we question whether this was an appropriate first topic for the response team to examine. OpenBSD was proposed and discarded, primarily due to the IT staff's unfamiliarity with the operating system. One member of the staff was familiar with Ubuntu, had a working Ubuntu installation (an experiment to support a new authentication infrastructure) and argued for this option. The IT staff has high turnover, so there was no RHEL expert currently employed and there were no individuals present who were capable of competently comparing RHEL and Ubuntu. Lacking any quantitative comparisons, no strong opposing voices emerged, and the Ubuntu motion carried.

Discussion moved on to the user directory and authentication system. The existing mechanism was based on NIS. As we mention above, one member of the IT staff had a pre-built LDAP server in place, so movement to LDAP was quickly agreed upon, especially because this provided a reason for the Ubuntu switch.

The agreement of those in the meeting was that a new network, independent of the existing network, had to be created, and each account had to be re-created with fresh authentication credentials (passwords, SSH keys) in the new network. Since it was finals week, most machines were under heavy use. An underused 8-machine cluster was proposed as a testbed for the deployment, and the group agreed that that cluster should become the testbed for the Ubuntu rollout.

Now that an overall plan was in place, the next question was one of prioritization. Since it was possible that the attack had been an insider attack (perhaps aimed at gleaning final exam information), the highest priority was to build clean Ubuntu images for the faculty. Thus, the faculty and finals remained the first critical concerns.

The December 17 meeting then broke up, and the IT staff began work. The first public disclosure of the attack happened one hour later when the IT manager emailed all faculty and PhD students informing them of the intrusion. All faculty passwords were to be changed.

On December 18, all PhD students teaching classes were informed that they would have to undergo the same procedure outlined for faculty the previous day. The department was also notified about the impending staffing shortage; half the IT workforce were leaving for jobs in the finance industry at the end of the year (in three days). Faculty cell-phone numbers were requested so staff could text message them new passwords. Installation of Ubuntu and LDAP on the test cluster began.

Students and staff then departed for the holiday break. The IT staff returned on December 27, and a Ubuntu

rollout on another computer cluster began (this time, a general-purpose lab). The next day, Solaris machines were upgraded to Solaris 10. On January 8, all guests and visitors were moved to the new system.

3.5 March 2008 attack

The March 2008 attack was detected by a member of the IT staff who noticed a new account named `mysqld` with root privileges on an important web server. Examining the contents of the home directory of this account showed several interesting files.

1. `.bash_history` containing what is probably a partial record of the attacker's behavior.
2. `ali.txt` containing the results of an NMAP scan for port 5555 (`freeciv`) across a /16 network.
3. `bot.pl` An IRC-based bot engine.
4. `dos.pl` A simple denial-of-service engine.
5. `xpl.c` Source code for the vmsplice Linux local privilege escalation exploit.

The `mysqld` account appeared in the `lastlog` history, along with the attacker's source IP address. Searching for that address in the Apache web server logs indicated that the attacker had repeatedly requested several files in a directory containing a common PHP web application, which was several revisions out of date, with remote exploits in the wild. The attacker added a copy of the `nsTView` remote web administration tool to the web app directory, leaving it set up with the default password.

The Apache logs also indicated that the attacker had downloaded a file he had created called `secret.txt`, containing the username and password for the web application's MySQL database, and the IP address for the remote host on which the database was running. Unfortunately, logging was disabled on the MySQL database, so investigations are limited in that direction. It is unknown whether the attacker ever connected to that database, or used one of several MySQL privilege escalation attacks to examine any of the other databases on that server.

We do note that, given the age of the web-application exploit, we believe that it is unlikely this is the first attacker to come in through this vulnerability. Furthermore, the `nsTView` remote web administration tool was using a default password, so multiple attackers may have come in through that route.

3.6 March 2008 response

The response to the attack began by removing `mysqld` from `/etc/passwd` in order to disable it. The MySQL server daemon was shut down shortly thereafter. The owner of the vulnerable web application was then contacted and it too was shut down. These responses were performed quickly – within two hours of the attack first being detected – and then the response turned to a policy discussion. What architecture and policy changes need to take place to prevent such attacks in the future? Several alternatives have been discussed, including undertaking a manual review of all web applications, prohibiting web applications entirely, making patching the mandatory responsbility of users running web applications, and moving the web infrastructure to a "read-only" style web site that is periodically refreshed from virtual machine snapshots. Users remain responsible for checking that their software is patched.

4 Incident Analysis

We next highlight some of the key decisions, discuss why they were not based purely on technical considerations, and suggest research directions aimed at helping automate and ease the process of decision making and reasoning under uncertain beliefs and knowledge. Note that our purpose is not to pass judgment on a particular decision by labeling it good or bad: the central goals of our analysis are to observe how non-technical factors influence decisions and to highlight what kinds of technical systems might help manage that influence.

4.1 Observations

Lesson 1: Cross-layer, anomaly-based intrusion detection seems valuable for detecting stealthy attacks. This type of detection is far more comprehensive than system call sequence monitoring and involves the fusion of alert streams from multiple levels of system abstraction.

All three attacks were discovered manually through symptoms and side–effects of each attacker's activities rather than traditional intrusion sensors like Snort or a commercial anti-virus product. At the time, the network did not employ a traditional network IDS, and little in the way of automated detection beyond some syslog monitoring scripts, but neither was the active attack sequence something detectable by a network intrusion detetion or a desktop anti-virus software system. In the March 2007 attack, abnormal kernel activity prompted an investigation by an IT staff member. In the December 2007 attack, crashes noticed by the Graphics research group led to the eventual discovery of the rootkit. The March 2008 attack was noticed by an IT staff member discovering a new privileged user account by accident — prompted by a trouble ticket filed by a senior professor asking why some standard mount points were failing.

This situation suggests that alert and educated IT staff and users are critical to uncovering stealthy attacks. We acknowledge that the sample size of incidents is small and purposefully focused on extensive intrusions (rather than well-known worm infection attempts). This lesson should be taken as a call to focus on creating anomaly sensors that span multiple levels of a system. For example, a system that correlates a user's inability to mount their regular partitions with anomalous network or host traffic can help build evidence for a comprehensive anomaly. The research challenge here is to move beyond AD techniques that rely solely on various flavors of system call sequence modeling.

Lesson 2: Staff do not have the luxury of complete forensics.

From an end-user viewpoint, this lesson was rather surprising at first, perhaps because we believe computer systems to be more flexible than they are in reality. Although we knew that undertaking an effective forensics process is challenging, we were surprised at the nasty dilemna of trying to analyze a host that one also wants to keep running. A tension exists between short-term operational demands to keep services running and long-term demands from the ISP to keep a network clean. Disks and machines have to be kept in use; we suspect that many organizations lack the luxury of taking them offline for extensive cleanroom analysis. Hot swappable and mirrored disks do offer a way to keep a machine online while also looking at a snapshot of the current content, but not all organizations can afford this type of redundancy for all their machines.

For example, if a critical server has been infected, the IT staff might decide that it is more important to quickly reinstall the server and restore normal operation than to analyze the malware in any depth. But while operational demands are important, the forensic analysis they preclude might reveal information which ultimately proves more critical still – perhaps it establishes that the infected server also infected other servers, or it might show that a compromised administrator account was the initial source of the intrusion, meaning that a reinstallation alone will not solve the overall problem.

An ISP often imposes constraints on real-time analysis of infected machines. IT staff may wish to analyze an infected machine's traffic to see if any other machines are communicating with it (and thus might be infected). But ISPs are often more concerned with limiting damage caused by an infected host. They will sometimes insist upon removing it from the network immediately, especially in academic environments, where the university is directly responsible for most hosts on the network. Large public ISPs may be less demanding to match their reduced liability.

Lesson 3: Visualizing a decision surface can help inform overall strategy and planning.

After detecting an incident or intrusion, it is difficult to immediately identify and execute the appropriate next steps; a staff is effectively in the middle of diagnosis. Staff may be torn between a number of actions, including continuing diagnosis and forensic efforts, fixing the immediate problem or small–scale symptoms of an attack (turn off a particular service, unplug a particular machine, remove a login entry from `/etc/passwd`), and fixing the larger–scale symptoms or root causes of an intrusion.

In the medium to long term, staff members needed a system that could direct the implementation of the solutions they had arrived at. To a certain extent, such a system includes standard "trouble ticket" or issue tracking software. In contrast to such an "obvious" tool, the technology that the staff actually used to plan out recovery activities for the December 2007 attack included a whiteboard and a marker. The whiteboard was inadvertently erased. The marker remains at large. Interestingly enough, usability research on display–centered group activities has found that displays are important in the planning stages of the activity, but grow progressively less useful as the plan is enacted [12].

In the short term, staff members needed a system that could direct planning activities by giving them a feel for the magnitude and location of various pitfalls (whether human or technical in nature). We suggest the concept of a decision surface composed of **process clocks** (a visual representation of task complexity using an estimate of task difficulty to shade in a graph node) as one way to achieve this high level view of the difficulty of the terrain ahead. We have found that standard decision trees and swim-lane diagrams are not quite appropriate for this goal, but we are left without any ready alternatives.

Decisions, and the reasons for making them, can be difficult to articulate and defend. Describing a decision making process can leave one lost for words — sometimes elements of the decision were based on intuition, flashes of inspiration or emotion, a complex sequence of data analysis, or deep contemplation and personal reflection. However hard it is to describe the process of making a decision, we have found that visualizing the elements of a decision is even harder. One of our main inspirations for writing this paper was the lack of a way for our system administrator to assess — at a quick glance — the difficulty of the terrain ahead of her, including parts of the decision surface where human and technical factors would conspire to greatly increase (or even decrease!) the complexity of the available alternatives. We have asked a number of our colleagues for their best method of visualizing a decision, and we have repeatedly drawn blanks. We consulted Edward Tufte's work[2] in hopes of

gaining some insight into visualizing the elements of information involved in a decision, but most information visualization principles did not seem directly applicable to this problem of visualizing a *process* (rather, a collection of processes).

As a result, we are attempting to define a model for visualizing a decision surface that would take into account the properties we observed to be important in guiding the process of network intrusion recovery: amount of human involvement, estimated effort for task completion, ordering dependencies of tasks, potential disruptions. We start by seeking to construct what we call a *decision surface*: a two dimensal plane akin to topographical maps projecting three dimensions onto a flat surface. The peaks, valleys, and plains of a decision surface convey at a glance where difficult or complex decision points lie. Knowing how to compose a decision surface, however, especially in light of future attacks, is a difficult exercise.

Lesson 4: Rapidly setting and executing a diagnosis and recovery agenda requires an unobtrusive, pervasive incident recording and modeling system that can help manage cognitive traps like availability bias and the shortcomings of human memory. Since human memory and recall is far from perfect, multiple points of view supply sometimes conflicting details of attacks and do not assist efforts in forensics, auditing, or planning for the next attack. Recovery graphs may provide one way to encode intrusion scenarios and the human response to them for later auditing.

The crucial first minutes after an intrusion discovery, in which there is no complete information about the attacker's entry point(s), history of actions, short and long-term intent, or current level of activity, hold the potential for panic, an overwhelming amount of data to analyze, and a paralyzed thought process. Involving too many people in the decision–making process after a serious intrusion is discovered can distract the person in charge. The hallway discussion on 17 December involved multiple people, ideas, and proposals. The system administrator involved with our case study achieved a certain level of success at repairing the network only because she was able to rapidly sift through the different proposals that the team members articulated.

Decision making at this point should be aided by automated processes that help manage the signal-to-noise ratio; in studies on decision–making, the manner in which information is organized often appears more important than simply getting increased amounts of information [20].

Furthermore, during our interviews, we observed that details of the attacks and the responses often differed wildly between individuals. Individuals often disagreed on dates — one person confused an attack from March 2007 with one from May 2006 and provided a mixture of details from both. In other cases, individuals presented radically different reports on which actions were taken. Two members of the IT staff disagreed on the date and method of detection of the December 2007 attack, while another viewed it as a continuation of the March 2007 attack. Without a coherent view of the state of the network, it is difficult for staff to make informed decisions to guide the attack response. One suggestion is that a staff member be tasked to record all the actions of a recovery process, but such a role can prove problematic for organizations that have staff shortages and tight budgets.

Even though researchers have proposed work on attack scenarios and attack trees [18, 22], relatively little attention has been paid to analyzing the process of a response. Automatically increasing the rate and types of events logged after an intrusion is discovered and the recovery process is started can assist efforts to revise a disaster recovery plan. More logging can make sure that key decisions are clearly recorded and not subject to human recollection of events occurring during a stressful time of rapid change and high rates of information. This type of recording is substantially different than ensuring that `/var/log/messages` collects more OS–level events. We propose the concept of *recovery graphs* to help capture and encode the sequences of events following the start of a recovery effort.

The lack of a human-centered post-intrusion journaling system suggests that research to design and develop new systems that record human–level events, judgments, recollections, and intentions is needed. Such systems must interact with humans seamlessly: they cannot place an additional burden on already-busy personnel. Categorizing, tagging, and cross-referencing events and information generated during the post-intrusion recovery process can help form a coherent view of what has happened and is happening to the network.

Lesson 5: Designing and maintaining a disaster recovery plan can aid recovery efforts, but the plan must be continuously — not periodically — updated.

The IT staff did not have *a priori* knowledge of what procedures should be enacted to combat or rectify the intrusion or to process and prioritize information about the incident. While the lack of a disaster recovery plan is a major operational shortcoming, disaster recovery plans alone are not a panacea. Like any proactive defense method, the plan may be incomplete, outdated, or unlikely to work given the current personnel. For example, the IT manager in our case study faced a critical personnel shortage due to events unrelated to the intrusion: half the staff was leaving for new jobs in a matter of days.

A disaster recovery plan must constantly evolve. Each new attack, vulnerability, or patch affects the recovery details. Similarly, employee turnover, improved employee skill set, and application deployment require

modifications to the plan. The question of how often to update the disaster recovery plan is a risk analysis and assessment task that must balance the needs of the staff to accomplish everyday system administration tasks against spending an inordinate amount of time planning for disasters that might never occur.

The open research question here is how personnel changes, catalogs of personnel skill, and lists of resources, sensors, countermeasures, toolsets, and inventory can drive an *automated* (and potentially real-time) update of the disaster recovery plan. We recognize that this research goal is rather ambitious (some readers have called it unrealistic — although existing pervasive recording systems [8] indicate otherwise), but we stress that this type of problem is precisely where the research gap is: little or no work in this space looks at ways to combine both humans and computers into a cohesive system where the computational elements are responsive, proactive, and transparent to the human as they go about their main tasks. In our minds, such a research direction is new and exciting, especially in a subfield where the bulk of the research looks at tweaking IDS parameters, considering an endless array of new features, or slicing up botnets in a variety of ways.

Nevertheless, the need to improvise can lead to creative solutions. For example, one of the most interesting countermeasures taken by the system administrator in the December 2007 attacks was to find an alternative distribution channel for new login credentials. The administrator sent text messages to the bulk of the user population with their new account password. This side channel is inexpensive (we estimate ten cents per message for roughly one thousand users), and it served quite nicely to distribute authentication material to users who were physically dispersed over the winter break.

Lesson 6: Decisions about appropriate technology shifts are driven by informal personal inclinations rather than quantitative (or even qualitative) comparisons of system properties.

Making changes to a complex and corrupted infrastructure requires (besides a quality analysis of the intrusion) a good understanding of the benefits offered by selecting one technology over another. For example, when the staff discussed whether to change computing platforms from RHEL to Ubuntu, the decision was made without any point–by–point comparison of the *security* benefits of either system. Although a question was raised about whether or not Ubuntu incorporated SELinux by default, as RHEL does, it was neglected (a symptom of the need for a recording system). The staff expressed comfort with Ubuntu's package management software and indicated that one staff member had already prototyped an Ubuntu system that would support stronger authentication. While good package management software can greatly ease the job of system administration, we feel that it is not the primary or only factor in a security–related decision. In this instance, however, the intrusion presented an opportunity for the IT staff to increase the security of the system.

Note that this decision represents an astoundingly rapid shift; even though the underlying platform is Linux, the actual delta is significant (placement of system files and scripts, customizations and patches to the kernel, *etc.*). Even such a minor shift stretches the limits of possibility; for example, deciding to switch the infrastructure to Windows or *BSD would not have been possible in the same amount of time the RHELv4/Ubuntu shift was accomplished.

This lesson points to the need for research into models or techniques to help estimate or otherwise provide some quantitative measure of how the defense posture will be affected after choosing to implement technology X over technology Y. Techniques like attack graphs [18, 22] and event-correlation [17] may help by focusing attention in important places, but at that point we need to begin the process of helping to make recovery decisions.

In this case study, the IT staff did not perform even a cursory examination of the release notes of the latest versions of the operating systems under consideration. While the circumstances and the time pressure demanded a quick decision, it would be best if the IT staff were not placed in such a bind to begin with. Providing systems that automated these types of comparisons and parameterizing them with the details of the intrusion or incident can assist staff efforts to make rational, informed, and *technical* decisions rather than strictly intuitive ones.

We can think of at least three research directions stemming from this lesson. It may be possible to use Natural Language Processing techniques to compare the release notes of the latest versions of two (or more) pieces of software for items that may impact the security posture of the organization. Second, a more realistic goal may be to create a system that data mines bug report databases and vulnerability mailing lists for items that are relevant to the security of the software systems under consideration. Finally, if the source of the intrusion can be traced to a weakness in a particular software package, it may be possible to work forward to predict other vulnerable components in that software [15].

4.2 Where Do we Go From Here?

Accounts discussing the trapping and tracking of attackers in improvised honeypots form part of the classic network security literature [25, 5]. Just as these accounts relate the first examples of a honeypot and computer forensics, the improvisation required in these early responses forecast exactly the plight of network administrators to-

day: when faced with a real attacker, decisions must be made quickly and accurately, and the decisions may conflict with the desires of other stakeholders. At the times of these early incidents, almost no tools existed to help trace hacker activity. Tools were improvised from the ground up, and their descendants and offshoots have become part of a standard set of tools. Now, network intrusion recovery faces an even larger challenge: create a suite of tools that take into account not only the engineering challenges of repairing a network, but also the human issues surrounding this process.

We have five specific suggestions for the construction of tools that could reasonably see use in the near term:

1. A way of visualizing complex decisions at a high level. A decision surface could help convey terrain information rather than just a branching factor of standard decision trees.

2. An unobtrusive, pervasive, and real-time activity monitor capable of efficiently and reliably recording both computer events and human actions during the recovery process.

3. A standard for encoding intrusion scenarios derived from the data captured by the above system, including the behavior of both human actors and computer systems in terms of the information structures they maintain and the sequences of actions they take. The keys to this standard are both fidelity and portability, so that these scenarios can be run on simulation infrastructures that employ, for example, different virtual machine hosts.

4. An environment for executing, analyzing, and reviewing these scenarios. For this environment, we can turn to recent work in the arena of computer game design [1] that focuses on the simulation of realistic crowds (rather than randomly milling zombies or predictably scripted bots) for a variety of purposes, including realistic storylines, evacuations from buildings or transportation vehicles in a crisis, and automated assessment of the usability and ergonomics of functional living or working space. This type of tool is useful for both post-mortem analysis to learn from the incident and to ensure that a recovery plan was fully enacted.

5. A toolset for automatically analyzing relevant security properties of alternative solutions. We do not see a panecea here; rather, it is likely that a collection of tools, each specialized to assessing the quality of a particular type of solution, is appropriate.

In each case, these tools help a team of administrators remove guesswork and uncertainty from the process of recovery. We also see a need for a way to input organizational changes to drive changes in a disaster recovery plan, but as we relate above, this task may prove to be too challenging, even if we manage to cobble together some combination of MindMap[3] and a trouble-ticket system.

5 Discussion

This paper has benefited greatly from both formal and informal feedback and reviews. Here, we would like to address some of the meta-issues and high-level concerns that various readers have raised. Fundamentally, we see this paper as the start of a two-pronged effort: first, to formally document intrusion recovery scenarios and second, to create systems that help support intrusion recovery efforts or that streamline the process of intrusion recovery.

The most obvious shortcoming of this paper is that we examine a single organization. It is hard to assess if the same specific troubles affect other organizations, but from our experience and anecdotal evidence, this similarity seems to be the case, at least for academic networks as well as some corporate networks we are familiar with. Some readers have suggested that the nature of the network itself suggests an administrative staff unconcerned with security, and thus it provides an unreasonable organization to base a case study on. Given our direct experience with the personnel involved in these incidents, we believe this is an unfair criticism of their efforts. Strategic security adjustments *are* important to the staff, but so are the day to day struggles — on a tight budget — to keep an infrastructure with many diverse interests and stakeholders operational. Furthermore, at least one other system administrator purposefully and publically runs a network without firewalls [23], like our subject network. Therefore, we suggest that this network is in fact typical of academic-style, open-access networks, and we do not claim that this network is the ideal model for drawing conclusions about, for example, a highly sensitive military network. Nevertheless, recovering from an intrusion remains a common problem, and the travails of the least prepared of us can help even those who are most prepared understand the risks they face.

We anticipate documenting new incidents as well as incidents from other organizations. We are in contact with the technical staff of our institutions to help broaden the scope of this research. We intend to start an archive of structured encodings of these scenarios. Such an archive can support comparisons between organizations as they respond to similar incidents and chose different tradeoffs.

We recognize that incidents similar to the ones we cover in this paper occur every day in many organizations worldwide. Far from making the details we expose here mundane, this reality underscores the fact that this topic is of critical concern. Furthermore, to the best of

our knowledge, no one is documenting these incidents in detail or examining how these details change over time. Descriptions of these incidents in the research literature are rare; we cited those that we could find.

Our belief is that the details of these incidents (and how organizations recover from them) are even more revealing and of as much interest as their high–level structure. Furthermore, since organizations have little incentive (in fact, they have potentially large legal and financial disincentives) to share the details of these incidents, academic research into methods of intrusion recovery remains uninformed and undirected. No concretely specified collection of intrusion recovery scenarios exist, and this lack leaves most discussions about the best way to recover from an attack at the level of hand–waving.

Other readers have suggested a variety of areas for further work and improvements, from performing a user impact survey to estimating the economic impact of the intrusions on the organization. We refrain from including this type of analysis precisely because there are no widely-accepted frameworks (although some nascent research proposals do exist [4]) for providing realistic, standardized estimates of costs for losses due to security incidents. Informal industry studies often hyperinflate their estimated costs to serve some agenda, be it marketing a particular security tool, the worth of their own survey, or to provide "evidence" that the problem is worth significant public or private investment. Our goal in this paper is simply to tease out how the technical and human complexities in specific, real-life scenarios interact — not to provide some exotic finanical estimation instrument, especially as none of the authors has any meaningful training in economics.

Finally, one aspect of intrusion recovery that we did not discuss is that of gathering forensic evidence to support criminal prosecution. The prevailing wisdom in this area is twofold. First, many attackers tend (or appear based on the attack source IP address) to be from jurisdictions outside of the US; as the organization we deal with is located in the US, it is unlikely that any such evidence would have been utilized in a criminal trial. Furthermore, many organizations hesitate to bring charges, because doing so requires that the incident become public knowledge. Neverthless, retaining log files and disk images of compromised machines can assist efforts to uncover a larger pattern of malicious activity. In any event, the IT staff was far more concerned with rebuilding the infrastructure and denying access to the intruder than preserving any chain of evidence (Section 4 discusses how IT staff find themselves in a bind when it comes to forensics).

6 Conclusion

Currently, repairing a network infrastructure after a serious intrusion is costly because cleanup is largely a manual process, and the complexity of information systems makes it difficult to automatically trace the extent of the attack. Furthermore, the psychological and sociological aspects of the problem are grossly understudied. Systems involve people, and their security decisions and risk assessments are often based on reasons that are not purely technical. The purpose of this case study is not to question whether the IT staff could have done a better job, or if the organization should have had a more robust network to begin with.

Instead, the lessons we should learn are that real security problems — those whose scope is sometimes too large to comprehend and deal with in any single research publication, are brushed aside as either too large to be interesting, or too close to human and organizational problems to be strictly "systems" security issues. With this case study, we hope to show that interesting possibilities for systems security research exist. Fundamentally, we think that human decisions should be assisted with automated methods that help filter and classify the available information. The problem of network intrusion recovery is a particularly thorny exercise in researching, designing, and creating usable security mechanisms.

Acknowledgements

We would like to thank our shepherd, Nicole Velasquez, for helping us resolve the issues and insightful comments raised by the reviewers. We deeply appreciate the cooperation and help of the IT staff that provides the subject of this paper. Theresa Menzel provided extensive feedback and anecdotal evidence from her experiences with intrusion incident handling. Locasto is supported in part by grant 2006-CS-001-000001 from the U.S. Department of Homeland Security under the auspices of the I3P research program. The I3P is managed by Dartmouth College. The opinions expressed in this paper should not be taken as the view of the authors' institutions, the DHS, or the I3P.

References

[1] BADLER, N., ALLBECK, J., ZHAO, L., AND BYUN, M. Representing and Parameterizing Agent Behaviors. In *Proceedings of Computer Animation* (June 2002), pp. 133–143.

[2] BHATKAR, S., DUVARNEY, D. C., AND SEKAR, R. Address Obfuscation: an Efficient Approach to Combat a Broad Range of Memory Error Exploits. In *Proceedings of the* 12^{th} *USENIX Security Symposium* (August 2003), pp. 105–120.

[3] CAMPOS, L., AND ALONSO-QUECUTY, M. L. Remembering a Criminal Conversation: Beyond Eyewitness Testimony. *Memory 14*, 1 (2006), 27–36.

[4] CARIN, L., CYBENKO, G., AND HUGHES, J. Quantitative Evaluation of Risk for Investment Efficient Strategies in Cybersecurity: The QuERIES Methodology. *IEEE Computer* (2008).

[5] CHESWICK, B. An Evening with Berferd, in which a cracker is lured, endured, and studied. In *Proceedings of the Winter USENIX Conference* (January 1992).

[6] CHESWICK, W. R., AND BELLOVIN, S. M. *Firewalls and Internet Security: Repelling the Wily Hacker*. Addison-Wesley, 1994.

[7] DUNLAP, G. W., KING, S., CINAR, S., BASRAI, M. A., AND CHEN, P. M. ReVirt: Enabling Intrusion Analysis Through Virtual-Machine Logging and Replay. In *Proceedings of the 2002 Symposium on Operating Systems Design and Implementation (OSDI)* (February 2002).

[8] GEMMELL, J., LUEDER, R., AND BELL, G. The mylifebits lifetime store. In *ETP '03: Proceedings of the 2003 ACM SIGMM workshop on Experiential telepresence* (New York, NY, USA, 2003), ACM, pp. 80–83.

[9] GRIZZARD, J. B., KRASSER, S., OWEN, H. L., DODSON, E. R., AND CONTI, G. J. Towards an approach for automatically repairing compromised network systems. In *Proceedings of 3rd IEEE International Symposium on Network Computing and Applications* (August 2004), IEEE, pp. 389–392.

[10] HAWKEY, K., MULDNER, K., AND BEZNOSOV, K. Searching for the Right Fit: Balancing IT Security Management Model Trade-Offs. *IEEE Internet Computing* (May/June 2008), 22–30.

[11] HILZINGER, M. Fedora: Chronicle of a Server Break-in. `http://www.linux-magazine.com/linux_magazine_com/online/news/update_fedora_chronicle_of_a_server_break_in`, March 2009. Linux Magazine.

[12] HUANG, E. M., MYNATT, E., AND TRIMBLE, J. P. Displays in the Wild: Understanding the Dynamics and Evolution of a Display Ecology. In *Proceedings of the* 4^{th} *International Conference on Pervasive Computing* (May 2006).

[13] KING, S. T., AND CHEN, P. M. Backtracking Intrusions. In 19^{th} *ACM Symposium on Operating Systems Principles (SOSP)* (October 2003).

[14] KURSAWE, K., AND KATZENBEISSER, S. Computing Under Occupation. In *New Security Paradigms Workshop* (September 2007).

[15] NEUHAUS, S., ZIMMERMANN, T., AND ZELLER, A. Predicting Vulnerable Software Components. In *Proceedings of the* 14^{th} *ACM Conference on Computer and Communications Security (CCS)* (2007).

[16] NEWSOME, J., AND SONG, D. Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software. In *Proceedings of the* 12^{th} *Symposium on Network and Distributed System Security (NDSS)* (February 2005).

[17] NING, P., CUI, Y., AND REEVES, D. S. Analyzing Intensive Intrusion Alerts Via Correlation. In *Proceedings of the 5th International Symposium on Recent Advances in Intrusion Detection (RAID 2002)* (October 2002).

[18] OU, X., BOYER, W. F., AND MCQUEEN, M. A. A Scalable Approach to Attack Graph Generation. In *Proceedings of the* 13^{th} *ACM Conference on Computer and Communications Security (CCS)* (October 2006).

[19] OZGIT, A., DAYIOGLU, B., ANUK, E., KANBUR, I., ALPTEKIN, O., AND ERMIS, U. Design of a log server for distributed and large-scale server environments.

[20] PAYNE, J. W., BETTMAN, J. R., AND JOHNSON, E. J. Behavioral Decision Research: A Constructive Processing Perspective. *Annual Review of Psychology 43* (1992), 88–131.

[21] PROVOS, N. Improving Host Security with System Call Policies. In *Proceedings of the* 12^{th} *USENIX Security Symposium* (August 2003), pp. 207–225.

[22] SHEYNER, O., HAINES, J., JHA, S., LIPPMANN, R., AND WING, J. Automated Generation and Analysis of Attack Graphs. In *Proceedings of the IEEE Symposium on Security and Privacy* (May 2002).

[23] SINGER, A. Tempting Fate. *USENIX login; 30*, 1 (February 2005), 27–30.

[24] SPAFFORD, E. H. The Internet Worm: Crisis and Aftermath. *Communications of the ACM 32*, 6 (June 1989), 678–687.

[25] STOLL, C. Stalking the Wily Hacker. *Communications of the ACM 31*, 5 (May 1988), 484.

[26] SUH, G. E., LEE, J. W., ZHANG, D., AND DEVADAS, S. Secure Program Execution Via Dynamic Information Flow Tracking. *SIGOPS Oper. Syst. Rev. 38*, 5 (2004), 85–96.

[27] WELLS, G. L., AND OLSON, E. A. Eyewitness Testimony. *Annual Review of Psychology 54* (2003), 277–295.

Notes

[1] `www.hotadmin.org`

[2] `http://www.edwardtufte.com/tufte/`

[3] `http://freemind.sourceforge.net/wiki/index.php/Main_Page`

Two-Person Control Administration: Preventing Administration Faults through Duplication

Shaya Potter, Steven M. Bellovin and Jason Nieh
Department of Computer Science
Columbia University
{spotter, smb, nieh}@cs.columbia.edu

Abstract

Modern computing systems are complex and difficult to administer, making them more prone to system administration faults. Faults can occur simply due to mistakes in the process of administering a complex system. These mistakes can make the system insecure or unavailable. Faults can also occur due to a malicious act of the system administrator. Systems provide little protection against system administrators who install a backdoor or otherwise hide their actions. To prevent these types of system administration faults, we created ISE-T (I See Everything Twice), a system that applies the two-person control model to system administration. ISE-T requires two separate system administrators to perform each administration task. ISE-T then compares the results of the two administrators' actions for equivalence. ISE-T only applies the results of the actions to the real system if they are equivalent. This provides a higher level of assurance that administration tasks are completed in a manner that will not introduce faults into the system. While the two-person control model is expensive, it is a natural fit for many financial, government, and military systems that require higher levels of assurance. We implemented a prototype ISE-T system for Linux using virtual machines and a unioning file system. Using this system, we conducted a real user study to test its ability to capture changes performed by seperate system administrators and compare them for equivalence. Our results show that ISE-T is effective at determining equivalence for many common administration tasks, even when administrators perform those tasks in different ways.

1 Introduction

As computing systems become more complex, they have also become harder to administer. From a security perspective, these complex systems create an environment that is easier for rogue users, be they inside or outside attackers, to make changes to the system that hide their malicious attacks. For instance, Robert Hanssen, an FBI agent who was a Soviet spy, was able to evade detection because he was the system administrator for some of the FBI's counterintelligence computer systems [26]. This allowed him to determine if the FBI had identified his drop sites and if he was the subject of investigation [5].

Insider attacks have long been known to be very difficult to address. Most approaches involve intrusion detection or role separation. However, both are ineffective against rogue system administrators who can replace the system module that enforces the separation or performs the intrusion detection. This attack vector was described over thirty years ago by Karger and Schell [13] and still remains a serious problem.

Even if administrators can be trusted not to be malicious, they must deal with software that is very complicated. Mistakes can be easy to make and hard to identify before they cause problems. These mistakes can affect both the stability of the system and its security. A mistake that takes down an important service can prevent the machine from being usable or further administered, and can even let malicious attackers access the machine with impunity.

There are several approaches for preventing and recovering from faults that creep into a system, including partitioning, restore points, and peer review. One of the most effective approaches is two-person control [1]. This can be provided by having two pilots in an airplane, requiring two keys for a safe deposit box, or running two or more computations in parallel and comparing the results for a fault-tolerant computer system. We believe this concept can be extended to address problems in system administration by leveraging virtualization to create duplicate environments.

Toward this end, we created the "I See Everything Twice" [10] (ISE-T, pronounced "ice tea") architecture. ISE-T provides a general mechanism to clone execution environments, independently execute computations that

modify the clones, and compare how the resulting modified clones have diverged. The system can be used in a number of ways, such as performing the same task in two initially identical clones, or executing the same computation in the same way in clones with some differences. By providing clones, ISE-T creates a system where computation actions can be "seen twice", applying the concept used for fault-tolerant computing to other forms of two-person control systems. There is a crucial difference though between our approach for using replicas and replicas as used in fault-tolerant computing. Our goal is to compare for equivalence between two replicas that may not be completely identical, rather than simply run two identical replicas in lock step and ensure they remain identical.

We apply ISE-T's principle to change the way we administer machines to provide two-person control administration. As ISE-T allows a system to be easily cloned into multiple distinct execution domains, we can create separate clone environments for multiple administrators. ISE-T can then compare the separate set of changes produced by each administrator for equivalence to determine if the same changes were made. By comparing the sets of changes for equivalence, ISE-T improves management by allowing it to be done in both a fail-safe and auditable manner.

In ISE-T, we force administrative acts to be performed multiple times before it is considered correct. Current systems give full access to the machine to individual administrators. This means that one person can accidently or maliciously break the system. ISE-T's ability to clone an execution environment creates a new way to administer machines to avoid this problems. ISE-T does not allow any administrator to modify the underlying system directly, but instead creates individual clones for two administrators to work on independently. ISE-T is then able to compare the changes each administrator performs. If the changes are equivalent, ISE-T has a high assurance that the changes are correct and will commit them to the base system. Otherwise, if it detects discrepancies between the two sets of changes, it will notify the administrators about the differences so that they can resolve the problem. This enables fail safe administration by enabling a single administrator's accidental errors to be caught, while also preventing a single administrator from maliciously damaging the system.

ISE-T leverages both virtualization and unioning file systems to provide the administration clones for each administrator. ISE-T is able to leverage both operating system virtualization techniques, such as Solaris Zones [18] and Linux VServer [20], as well as hardware virtualization such as VMware [24], to provide each administrator with an isolated environment in which to perform the changes. ISE-T builds upon DejaView [14], leveraging union file systems to provide a layered file system that is able to provide the same initial file system namespace in one layer, while capturing all the system administrator's file system changes into a separate layer. This enables easy isolation of changes, simplifying equivalence testing.

ISE-T's approach of requiring everything to be installed twice blocks many real attacks. A single malicious system administrator can no longer install modules that create an intentional back-door to allow future access into the system. Similarly, they cannot unilaterally weaken firewall rules, nor create unauthorized accounts to allow others into the system.

ISE-T is admittedly an expensive solution, too expensive for many commercial sites. For high-risk situations, such as in the financial, government, and military sectors, the added cost can be acceptable if the risk is reduced. In fact, the two-person controls are already routine in those environments, ranging from checks that require two signatures to the well known requirement of requiring two people for any work involving nuclear weapons. However, we also demonstrate how ISE-T can be used in a less expensive manner by introducing a form of auditable system administration. Instead of requiring two system administrators at all times, auditable ISE-T captures all the changes performed by the system administrator in the same manner it uses for equivalence testing, but instead immediately saves it to an audit log while committing it to the underlying system. An audit can then be performed on the log to provided a higher level of assurance that the administrator was only performing the changes they claimed they were performing.

In a similar manner, ISE-T can be extended to train less experienced system administrators. First, ISE-T allows a junior system administrator to perform tasks in parallel with a more senior system administrator. While only the senior system administrator's solution will be committed to the underlying system, the junior system administrator can learn from how his solution differs from the senior system administrator. Second, ISE-T can help train junior system administrators by being extended to provide an approver mode. In this mode, a junior system administrator will be provided administration tasks to complete. However, instead of the changes being committed directly, they will be presented to the senior system administrator who can approve or disapprove of the changes, without being required to do the same actions in parallel.

We have implemented an ISE-T Linux prototype without requiring any source code changes to the underlying kernel or system applications. To evaluate its ability to do equivalence testing, we conducted a user study to determine ISE-T's ability to efficiently capture administration changes through its layered file system, as well as to

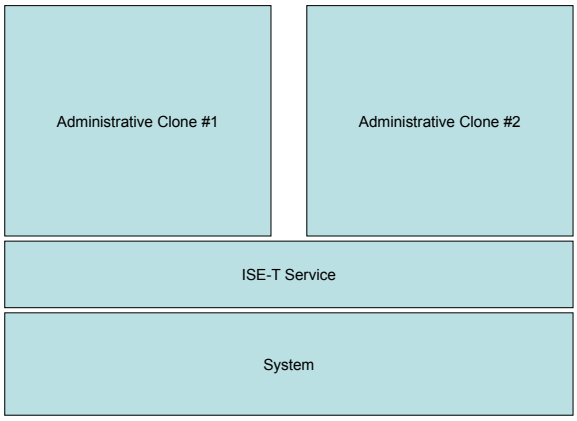


Figure 1: ISE-T Usage Model

compare the environments of the multiple administrators for equivalence. Our results demonstrate that ISE-T is effective at determining equivalence for many common administration tasks even when administrators perform those tasks in different ways. Furthermore, we demonstrate that ISE-T is able to easily show the differences that occur when the actions are not performed equivalently, in what situations the actions cannot be performed equivalently, as well as ISE-T's ability to detect malicious administration changes.

2 Usage Model

Systems managed by ISE-T are used by two classes of users, regular unprivileged users and the privileged system administrators who manage the machines. ISE-T does not change how regular users interact with the machine. They are able to install any program into their personal space, as well as run any program on the system, including regular programs and programs with special privileges, such as `setuid` UNIX programs that raise the privileges of the process on execution. This allows regular users to execute programs such as `passwd` to change their passwords.

However, ISE-T fundamentally changes the way system administrators interact with the machine. In a regular system, when administrators want to perform maintenance on the machine, they will leverage their ability to execute arbitrary programs with administrative privileges. This can be accomplished by executing a shell with the privilege so that they can execute arbitrary commands with ease, or by leveraging a program such as `sudo` that will just execute the arbitrary programs itself that way. In these systems, administrators are able to modify the system in a direct manner, change files, and execute programs and have those changes occur directly.

As ISE-T prevents system administrators from executing arbitrary programs with administrative privileges, this model cannot be directly used in a system managed by ISE-T. Instead, ISE-T provides a new model as shown in Figure 1. Instead of administering a system directly, ISE-T creates administration clones. Each administration clone is fully isolated from each other and the base system. ISE-T instantiates an administration clone for each administrator to perform the administrative acts within. Once both administrators are finished, ISE-T compares the clones for equivalence and commits the changes if the clones pass the equivalence test. As opposed to a regular system, where the administrator can interleave file system changes with program execution, in ISE-T only the file system changes get committed to the underlying system. Therefore ISE-T requires administrators to use other methods if they require file system changes and program execution to be interleaved on the actual system, such as for rotating log files or to do exploratory changes in order to diagnose a subtle system malfunction.

To allow this, ISE-T provides a new `ise-t` command that is used in a manner similar to `su`. Instead of spawning a shell on the existing system, `ise-t` spawns a new isolated container for that administrator. This container contains a clone of the underlying file system. Within this clone, the administrators can perform generic administrative actions, as on a regular system, but the changes will be isolated to this new container. When the administrators are finished with the desired administration changes, they exit their new container's shell, much as they would exit a root shell; the container itself is terminated, while its file system remains around.

ISE-T then compares the changes each administrator performed for equivalence. ISE-T performs this task automatically after the second administrator exits his administration session and notifies both of the administrators of the results. If the changes are equivalent, ISE-T automatically commits the changes to the underlying base system. Otherwise, ISE-T notifies the administrators of the file system discrepancies that exist between the two administration environments, allowing the administrators to recreate their administration environments and correct the discrepancies.

Command	Description
`ise-t new`	Create an administration environment
`ise-t enter`	Enter administration environment
`ise-t done`	Ready for equivalence testing
`ise-t diff`	Results of a failed equivalence test

Table 1: ISE-T Commands

As ISE-T only looks at file system changes, this can prevent it from performing administrative actions that just affect the runtime of the system. In order to han-

dle this, ISE-T provides a raw control mechanism via the file system, as well as enabling itself to be integrated with configuration management systems. First, ISE-T's raw control mechanism is implemented via a specialized file system namespace where an administrator can write commands. For instance, if the administrators want to kill a process, stop a service or reboot the machine, those actions performed directly within their administration container will have no affect on the base system. Some actions can be directly inferred from the file system. For instance, if the system's set of startup programs is changed, by having a file added, removed or replaced, ISE-T can infer that the service should be started, stopped or restarted when the changes are committed to the underlying system. However, this only helps when one is changing the file system. There are times when administrators will want the services stopped or restarted without modifying the file system of the system. Therefore, ISE-T provides a defined method for killing processes, stopping and starting services and rebooting the machine using files the administrator can store on the local file system. ISE-T provides each administrator with a special `/admin` directory for performing these predefined administrative actions.

For example, if the administrator wants to reboot the machine, they create an empty `reboot` file within the `/admin` directory. If both administrators create the file, after the the rest of their changes are committed to the system, it will reboot itself. Similarly, the administrators can create a `halt` file to halt the machine. In addition, the `/admin` directory has `kill` and `services` subdirectories. To kill a process, administrators create individual files with the names of the process identifiers of processes running on the base system that they desire to kill. Similarly, if a user desires to stop, start, or restart a `init.d` service, they can create a file named by that service prefixed with `stop`, `start` or `restart`, such as `stop.apache` or `restart.apache` within the `services` directory to have ISE-T perform the appropriate actions when the changes are committed to the base system. The files created within the `/admin` directory are not committed to the base system; they are only used for performing runtime changes to the system.

However, many systems already exist to manage systems and perform these types of tasks, namely configuration management systems, such as `lcfg` [2]. At a high level, configuration management systems work by storing configuration information on a centralized policy server that controls a set of managed clients. In general, the policy server will contain a set of template configuration files that it uses to create the actual configuration file for the managed clients based on information contained within its own configuration. Configuration management systems also generally support the ability to run predefined programs, scripts and execute predefined actions on the clients they are managing.

When ISE-T is integrated with any configuration management system, it no longer manages the individual machines. Instead of the managed clients being controlled by ISE-T, the configuration policy server is managed by ISE-T directly and the clients are managed directly by the configuration management system. This provides a number of benefits. First, it simplifies the complexity of comparing two different systems, as ISE-T can focus on the single configuration language of the configuration management system. Second, configuration system already have tools to manage the runtime state of their client machines, such as stopping and starting services and restarting them when the configuration changes. Third, many organization are already used to using configuration management systems; by implementing ISE-T on the server side, they can enforce the two-person control model in a more centralized manner.

3 ISE-T Architecture

To enable the two-person administrative control semantic, ISE-T provides three architectural components. First, as the two administrators cannot administer the system directly, they must be provided with isolated environments in which they can perform their administrative acts. To ensure the isolation, ISE-T provides container mechanisms that allow ISE-T to create parallel environments that are based on the underlying system that is being administered. This allows ISE-T to fully isolate each administrator's clone environment from each other and from the base system.

Second, we note that any persistent administrative action has to involve a change to the file system. If the file system is not affected, the action will not survive a reboot. Whereas some administrative acts only affect the ephemeral runtime state of the machine, the majority of administrative acts are of a more persistent nature. Therefore, to allow ISE-T to create two-person administrative control, the file system is a central component. ISE-T provides a file system that can create branches of itself as well as isolate the changes made to it. This enables the easy creation of the clone containers, as well as enabling the easy comparison of the changes performed to both environments.

Finally, ISE-T provides the ISE-T System Service. This service instantiates and manages the life-times of the administration environments. It is able to compare the two separate administration environments for equivalence to determine if the changes performed to them should be committed to the base system. ISE-T's System Service performs this via an equivalence test that compares the two administration environment's file sys-

tem modifications for equivalence. If the two environments are equivalent, the changes will be committed to the underlying base system. Otherwise, the ISE-T System Service will notify the two administrators of the discrepancies and allow them to fix their environments in the appropriate fashion.

3.1 Isolation Containers

ISE-T can leverage multiple different types of container environments, depending on the requirements of the administrators managing the system. In general, the choice will be between hardware virtual machine containers and operating system containers. Hardware virtual machines, such as VMware [24], provide a virtualized hardware platform that a separate operating system kernel runs on and provides a complete operating system instance. Operating system containers, such as Solaris Zones [18], on the other hand, are just isolated kernel namespaces running on a single machine.

For ISE-T, there are two primary difference between these containers. First, hardware virtual machines allow the administrators to install and test new operating system kernels as each container will be running its own kernel. Operating system containers, on the other hand, prevent the administrators from testing the underlying kernel, as there is only one kernel running, that of the underlying host machine. Second, as hardware virtual machines require their own kernel and a complete operating system instance to be started up, they take a significant amount of time to create the administration clones. On the other hand, operating system containers can be created almost instantly, allowing the administrators to quickly perform their actions. As both types of containers have significant benefits for different types of administrative acts, ISE-T supports the ability to use both. For most actions, administrators will prefer to use operating system containers, while still enabling them to get a complete hardware virtual machine when they desire to test kernel changes.

When ISE-T is integrated with a configuration management system, ISE-T does not have to use any isolation container mechanism at all, as the configuration management system already isolates the administrators from the client system. Instead, ISE-T simply provides each administrator with their own configuration management tree and let each individual administrator perform the changes.

3.2 ISE-T's File System

To support its file system needs, ISE-T leverages the ability of some file systems to be branched. Unlike a regular file system, a branchable file system can be snapshotted at some point in time and branched for future use. This allows ISE-T to quickly clone the file system of the machine being managed for both clone administration environments. As each file system branch is independent, this allows ISE-T to capture any file system changes in the newly created branch, by comparing the branch's state against the initial file system state. Similarly, ISE-T can then compare the set of file system changes from both administration clones against each other for equivalence.

However, while a classical branchable file system allows one to capture the changes, it does not allow one to efficiently discover what has changed, as the branch is a complete file system namespace. Iterating through the complete file system can take a significant amount of time, as well as place a large strain on the file system and decrease system performance. To allow ISE-T to use a file system efficiently, it must provide two features. First, it must be able to duplicate the file system to provide each administrator with their own unique and independent file system to perform their changes on. Second, it must provide a way to easily isolate the changes each administrator makes to the file system to easily test the changes for equivalence. To meet these requirements, ISE-T creates layered file systems for each administration environment, where multiple file systems can be layered together into a single file system namespace for each environment. This enables each administration environment to have a layered file system composed of two layers, a single shared layer that is the file system of the machine they are administrating, as well as a layer that will contain all the changes the administrator performs on the file system.

To support the creation of the layered file system, ISE-T has to solve a number of file system related problems. First, it must support the ability to combine numerous distinct file system layers into a single static view. This is equivalent to installing software into a shared read-only file system. Second, as users expect to be able to interact with the layered file system as a normal file system, such as by creating and modifying files, ISE-T has to enable the layered file system to be fully modifiable. In a related vein, the third problem ISE-T has to solve is that end users should also be able to delete files that exist on the read-only layer. However, end users should also be able to recover the deleted files by reinstalling or upgrading the layer that contains the deleted. This is equivalent to deleting a file from a traditional file system, but reinstalling the package that contains the file to recover it.

To solve these problems, ISE-T leverages union file systems. Unioning file systems enable ISE-T to solve the first problem as they allow the system to join multiple distinct directories into a single directory view, as shown in Figure 2. These directories are unioned by layering directories on top of one another. For example, when two directories are unioned together, one directory contain-

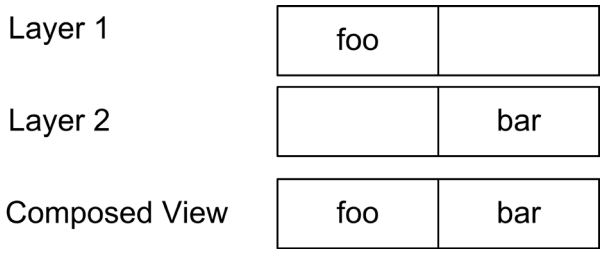


Figure 2: Unioning Namespaces

ing the file `foo` and the other containing the file `bar`, the unioned directory view would contain both files `foo` and `bar`. To provide a consistent semantic, most union file systems only allow one layer, namely the topmost to have files added to it. At the same time, if a file that already existed is modified, the union file system changes the underlying file directly, in whatever layer of the union it existed previously.

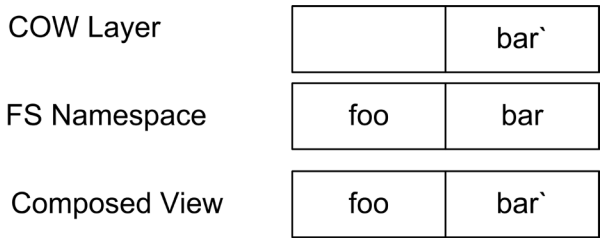


Figure 3: COW functionality

To solve the second problem, union file system can be extended [27] to enable them to assign properties to the layers, defining some layers to be read only while others can be read-write. This results in a model that borrows from copy-on-write (COW) file systems, where a modifying a file on a lower read-only layer will cause it to be copied to the topmost writable layer, as shown in Figure 3. For instance, in the above example, a blank cow writable layer can be layered on top of a read only layer containing `foo` and `bar`. If, in the course of usage, file `bar` get modified to `bar`` it will be *copied up* to the top most layer before the modification occurs. When a file is created or modified, it is written to the private read-write layer enabling the layered file system to be differentiated through file system changes.

This layering model also provides a semantic that directory entries located at higher layers in the stack obscure the equivalent directory entries at lower levels. Continuing the example, both layers now contain the file `bar`, but only the top most layer's version of the file is visible. To provide a consistent semantic, if a file is deleted, a white-out mark is also created on the top most layer to ensure that files existing on a lower layer are

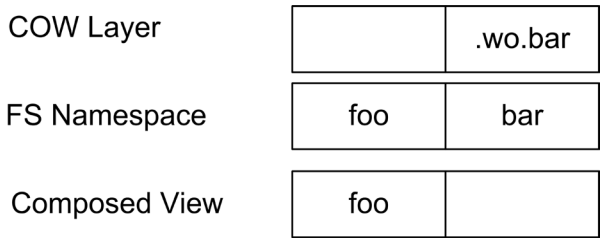


Figure 4: White-Out Support for Deletion

not revealed, as shown in Figure 4. Now, if the file `bar` were deleted, it would not allow the `bar` on the lower layer to be revealed. The white-out mechanism enables obscuring files on the read-only lower layers, simply by creating white-out files on the topmost layer.

ISE-T's layered file system provides the ability for multiple independent views of a file system to be in an active modifiable state at the same time, while confining each view's modifications to itself by providing each file system with an independent COw layer. To provide a simple example, imagine one has a directory that one wants to branch into two distinct views. This implies that processes operating in one view would be able to modify the files, without the changes causing any effect in the other view, and vice versa. This model can simply be implemented by ISE-T with the above union file system semantic. ISE-T creates two distinct views of the directory by creating two distinct ISE-T branched file system mounts. Since all modifications will cause files to be copied to the top most directory, it enables one to simply contain each views modifications into its own space. Finally, as each COW layer isolates the changes that were performed to each file system, ISE-T can easily determine which files it has to compare for equivalence.

3.3 ISE-T System Service

ISE-T's System Service has a number of responsibilities. First, it manages the lifetimes of each administrator's environment. When administration has to be performed, it has to setup the environments quickly. Similarly, when the administration session has been completed and the changes committed to the underlying system, it removes them from the system and frees up their space. Third, it evaluates the two environments for equivalence by running a number of equivalence tests to determine if the two administrators performed the same set of modifications. Finally, it has to either notify the administrators of the discrepancies between their two environments or commit the equivalent environment's changes to the underlying base system.

ISE-T layered file system allows ISE-T system's service to easily determine which changes each administra-

tor made, as each administrator changes will be confined to their personal layer of the layered file system. To determine if the changes are equivalent, ISE-T first isolates the files that it does not care about, and that will not be committed to the base system. This is currently limited to the administrator's personal files in their branch, such as shell history. Instead of just removing them, ISE-T saves them for archival and audit purposes. ISE-T then iterates through the files in each environment, comparing the file system contents and files directly against each other. If each administrator's branch has the equivalent set of file system changes, ISE-T can then simply commit a set to the base system. On the other hand, if the files contained within each branch, are not equivalent, ISE-T flags the differences and reports to each administrator what the differences are. The administrators can then confer with each other to ensure that they perform the same steps, so that they will create the same set of files to commit to the base system.

Determining equivalence can vary based on the type of file and what is considered to be equivalent. For instance, a configuration file modified by both administrators with different editors can visually appear to be equivalent, but can differ from each other if one uses spaces and another used tab characters. These files can be equivalent, as they would be parsed by applications in the same manner, but would be different when examined on a character by character level. However, there are some languages (e.g., Python) where the amount of whitespace matters; this can have a large effect on how the script executes. On the other hand, two files that have exactly the same file contents can have varying meta data associated with the file, such as permission data, extended attributes or even the multiple types of time data associated with each file. Similarly, some sets of files should not matter for equivalence, such as the shell history that recorded the steps the administrators took in their respective environments, and in general the home directory contents of the administrator in the administration environment. ISE-T prunes these files from the comparison, and never commits them to the underlying system.

Taking this into consideration, ISE-T's prototype comparison algorithm determines these sets of differences.

1. Directory entries which do not exist in both sets of changes are differences.
2. Every directory entry that does not have the same UID, GID, and permission set are different.
3. Every directory entry that is not of the same file type (Regular File, Symbolic Link, Directory, Device Node, or Named Pipe) are different

For directory entries that are of the same type, ISE-T performs the appropriate comparison.

- Device nodes must be of the same type
- Symbolic links must contain the same exact path
- Regular files must have the same size and the exact same contents

There are two major problems with this approach. First, this comparison takes place at a very low semantic level. It does not take into account simple differences between files that make no difference in practice. However, without writing a parser for each individual configuration language, one will not easily be able to compare equivalence. Second, there are certain files, such as encryption keys, that will never be generated identically, even though equivalent actions were taken to create them. This can be important, as some keys are known to be weaker and a malicious administrator can construct one by hand.

Both of these problems can be solved by integrating ISE-T with a configuration management system and teaching ISE-T the configuration management system's language. First, these systems simplify the comparison by enabling it to focus on the configuration management system's language. Even though most configuration management systems work by creating template configuration files for the different applications, these files are not updated regularly and can be put through the stricter exact comparison test. On the other hand, when ISE-T understands the single language of the configuration management system, it can rely on a more relaxed equivalence test. Second, configuration management systems already have to deal with creating dynamic files, such as encryption keys. A common way configuration management systems deal with these types of files is by creating them directly on the managed client machines. As ISE-T understand the configuration management system's language, the higher level semantics that instruct the system to create the file will be compared for equivalence instead of the files themselves. However, a potential weakness of ISE-T is in dealing with files that cannot easily be created on the fly and will differ between two system administration environments, such as databases. For instance, two identical database operations can result in different databases due to the saving of a time-stamp, or the simple reordering of updates on the database server.

4 ISE-T for Auditing

Whereas the two-person control model that ISE-T provides to system administration is useful for providing high assurance that faults are not going to creep into the system, its expense can make it unusable in many situations. For example, since the two-person control model

requires the concurrence of two system administrators on all actions, it can prevent timely actions from being taken if only a single administrator is available. Similarly, whereas the two-person control model provides a high degree of assurance for a price, it would be useful if organizations could get a higher degree of assurance than normal with little extra cost. To achieve these goals, we can combine ISE-T's mechanisms with audit trail principles to create an auditable system administration semantic.

In auditable system administration, every system administration act that is logged to a secure location so that it can be reviewed for correctness at some point in the future. The ISE-T System Service creates clone administration environments for the two administrators and can capture the state they change in order to compare them for equivalence. For auditable system administration, ISE-T's mechanism can also be used. The audit system prevents the single system administrator from modifying the system directly, but require the creation of a cloned administration environment where the administrator can perform the changes before they are committed to the underlying system. Instead of comparing for equivalence against a second system administrator, the changes are logged so that they can be used by an auditor at some point in the future as well as immediately committed to the underlying system. Audit systems are known to increase assurance that malicious changes are not performed, as the malicious person knows there's a good chance his actions will be caught. Similarly, depending on the frequency and number of audits performed, it can help prevent administration faults from persisting for long periods of time in the system. However, it does not provide as high assurance a model as can be provided by the two-person control system, as the administrator can use the fact that his changes are committed immediately to create backdoors in the system that cannot be discovered until later.

Auditable system administration needs to be tied directly to an issue-tracking service. This allows an auditor to associate an administrative action with what the administrator was supposed to accomplish. Every time an administrator invokes ISE-T to administer the system, an issue-tracking number is passed into the system to tie that action with the issue in the tracker. This allows the auditor to compare the results of what occurred with what the auditor expects to have occurred. In addition, auditable system administration can be used in combination with the two-person control system when only a single administrator is available and action has to be taken in a more immediate fashion. With auditing, the action can be performed by the single administrator, but can be immediately audited when the second administrator becomes available. This helps the system maintain its higher level of assurance when immediate action has to be taken by a single administrator.

5 Experimental Results

To test the efficacy of ISE-T's layered file system approach, we recruited 9 experienced computer users with varying levels of system administration experience, though all were familiar with managing their own machines. We provided each user with a VMware virtual machine running Debian GNU/Linux 3.0. Each VM was configured to create an ISE-T administration environment that would allow the users to perform multiple administration tasks isolated from the underlying base system. Our ISE-T prototype uses UnionFS [27] to provide the layered file system needed by ISE-T. We asked the users to perform the eleven administration tasks listed in Table 2. The user study was conducted in virtual machines running on an IBM HS20 eServer blade with dual 3.06 Ghz Intel Xeon CPUs and 2.5GB RAM running VMware Server 1.0. These tasks were picked as they are indicative of common administration tasks, as well as including a common way a malicious administrator would create a back-door in the system for himself.

Each task was performed in a separate ISE-T container, so that each administration task was isolated from the others, and none of the tasks depended on the results of a previous task. We used ISE-T to capture the changes each user performed for each task in its own file system. We were then able to compare each user against each other for each of the eleven tasks, to see if they performed equivalent modifications or where their modifications differed.

For every test, ISE-T prunes the changes that were done to remove files that would not affect equivalence since they would not be committed to the underlying file system, as described in Section 3.3. Notably, in our prototype, ISE-T prunes the `/root` directory which is the home directory of the root user, and therefore would contain differences in files such as `.bash_history` amongst others that are specific to how they went about performing the task. Similarly, ISE-T prunes the `/var` subtree to remove any files that were not equivalent. For instance, depending on how an administrator would administer the system and what tools one would use, different files would be created, for instance a cache of packages downloaded and installed via the `apt-get` tool versus being downloaded and installed manually. The reasoning behind this pruning is that the `/var` tree is meant as a read-write file system for per-system usage. Tools will modify it; if different tools are used, different changes will be made. However, one cannot prune the entire directory tree as there are files or directories within it that are necessary for runtime use and those

Category	Description	Result
Software Installation	Upgrade entire system via package manager	Equivalent
	Install official Rdesktop package	Equivalent
	Compile and install Rdesktop from source	Equivalent
System Services	Install SSH Daemon from package	Not Equivalent (Not Desired)
	Remove PPP package using package manager	Equivalent
Configuration Changes	Edit machine's persistent hostname	Equivalent
	Edit the inetd.conf to enable a service	Not Equivalent (Not Desired)
	Add a daily run cron job	Equivalent
	Remove an hour run cron job	Equivalent
	Change the time of a cron job	Equivalent
Exploit	Create a backdoor setuid root shell anywhere	Not Equivalent (Desired)

Table 2: Administration Tasks

changes have to be committed to the underlying file system. Therefore, only those changes that are equivalent were committed, while those that are different were ignored. ISE-T also prunes the `/tmp` directory as the contents of this directory would also not be committed to the underlying disk. Finally, due to the UnionFS implementation, ISE-T also prunes the whiteout files created by UnionFS if there is no equivalent file on the underlying file system. In many cases, temporary files with random names will be created; when they are deleted, UnionFS will create a whiteout file, even if there is no underlying file to whiteout. As this whiteout file does not have an impact on the underlying file system, it is ignored. On the other hand, whiteout files that do correspond to underlying files and therefore indicate that the file was deleted are not ignored.

5.1 Software Installation

For the software installation category, we had the users perform three separate tests that demonstrated different ways administrators install software into the system. These tests were to demonstrate that when multiple users install the same piece of software, as long as they install it in the same general way, the two installations will be equivalent.

To demonstrate this, the users were first instructed to install the rdesktop program from its Debian package. Users could choose to download the package by hand and install it via `dpkg`, they could use `apt-get` to download it and any unfulfilled dependencies, or use the `aptitude` front end amongst many ways to perform this task. Most users decided to install the package via `apt-get`, but even those who did not made equivalent changes. The only differences were those in pruned directories, demonstrating that installing a piece of prepackaged software using regular tools will result in an equivalent system.

Second, the users were instructed to build the rdesktop program from source code and install it into the system. In this case, multiple differences could have occurred. First, if the compiler would create a different binary each time the source code is compiled, even without any changes, one would have a more difficult time evaluating equivalence. Second, programs generally can be installed in different areas of the file system, such as `/usr` versus `/usr/local`. In this case, all the testers decided to install the program into the default location, avoiding the latter problem, while also demonstrating that as long as a the same source code is compiled by the same toolchain, it will result in the same binary. However, some program source code, such as the Linux kernel, will dynamically modify their source during build, for example to define when the program was built. In these cases, we would expect equivalence testing to be more difficult as each build will result in a different binary. A simple solution would be to patch the source code to avoid this behavior. A more complicated solution would involve evaluating the produced binary's code and text sections with the ability to determine that certain text section modifications are inconsequential. Again, in this case the only differences were in pruned directories, notaby the `/root` home directory to which the users downloaded the source for rdesktop.

Finally, we had the users upgrade the Debian stable system with all pending security updates. This was a more complicated version of the first test, as multiple packages were upgraded. Although differences existed between the environments of the users, the differences were confined to the `/var` file system tree and depended on how they performed the upgrade. This is because Debian provides multiple ways to do an upgrade of a complete system and those cause different log files to be written. As they all installed the same set of packages, the rest of the file system, as expected, contained no differences.

5.2 System Services

Our second set of tests involved adding and removing services: the users were instructed to install the ssh service and remove the PPP service. These tests were an extension of the previous package installation tests and were meant as a demonstration of how one would automatically start and stop services, as well as a demonstration of files we knew would be different and therefore fail equivalence testing.

For the first test, we instructed the users to install the SSH daemon. This test sought to demonstrate that ISE-T can detect when a new service is installed and therefore enable it when the changes are committed. This is demonstrated by the fact that in Linux systems, a System-V init script has to be added to the system to enable it to be started each time the machine boots. If the user's administration environment contains a new init script, ISE-T can automatically determine that the service should be started when this set of administration changes are committed to the base system. This test also sought to demonstrate that certain files are always going to be different between users if created within their private environment. This is demonstrated by the fact that the SSH host key for each environment is different. This is because it is created based on the kernel's random entropy pool that will be different for each user and therefore will never be the same if created in separate environment. A way to solve this would be not to create it within the private branch of each user, but instead have it be created after the equivalent changes are committed, for instance, the first time the service's init script is executed.

For the second test, we instructed the users to remove the PPP daemon. This test sought to demonstrate that there are multiple ways to remove a package in a Debian system and depending on the way the package is removed, the underlying file system will be different. Specifically, a package can either be `removed` or `purged`. When a package is removed, files marked as configuration files are left behind, allowing the packages to be reinstalled and have the configuration remain the same. On the other hand, when a package is purged, the package manager will remove the package and all the configuration files associated with it. In this case, the user's chose different ways to remove the package, and ISE-T was able to determine the differences for those that chose to remove or purge it.

5.3 Configuration Changes

Our third set of tests involved modifications to configuration files on the system and involved six separate tests. These tests could be subdivided into three categories. The first category was composed of simple file configuration changes. We first instructed the users to modify the host name of the machine persistently from `debian` to `iset`, which is accomplished by editing the `/etc/hostname` file. As expected, as this configuration change is very simple, all user modified the system's hostname in the exact same manner, allowing ISE-T to determine that all the systems were equivalent.

Next, we instructed the users to modify the `/etc/inetd.conf` file to enable the `discard` service. In this case, as the file is more free-form, their changes were not exact, and many were not equivalent. For example, some users enabled it for both TCP and UDP, while some users enabled it for TCP alone. Also, some users added a comment, while others did not. Whereas the first change is not equivalent, the second change should be considered equivalent, but this cannot be determined by a simple diff; one needs the ability to parse the files correctly to determine that they are equivalent, an ability our ISE-T prototype does not have. However, ISE-T was able to clearly report the differences that existed between the environments of users who performed this administration task differently.

The second set of tests involved setting up and removing cron jobs and was composed of three tests. First, we provided the users with a script in `/root` that we instructed them to install in the system in a manner so that it will be executed daily. In Debian there are two ways to have a cron job execute daily. The best way is to simply place the script in the `/etc/cron.daily` directory, as all the programs located within it are executed once a day. On the other hand, a user could use the `/etc/crontab` file and instruct it exactly what time each day to execute the program. We were looking for users to place the file in `/etc/cron.daily` which the large majority did; but one user who perhaps was not used to Debian modified `/etc/crontab` directly. Again, ISE-T was able to detect the difference when it existed and report the discrepancy. Similarly, we instructed users to remove a cron job that ran hourly. The cron job's script was already located in `/etc/cron.hourly` and not in `crontab`. All users deleted it from the `/etc/cron.hourly` directory, even the user who modified `/etc/crontab` in the previous test. Finally, we had an entry in `/etc/crontab` that executed a cron job at 4:35 AM every day, and instructed the users to change it to 5:35 AM. As expected, all users performed the same set of changes by modifying the `/etc/crontab` file.

5.4 Exploit

In our final test, we asked the users to create a backdoor shell into the system. This test demonstrates that ISE-

T will detect when one of the two system administrator makes a rogue administration change to the system, as the other administrator will not have made that same administration change. We did not specify how or where the shell should be created, and left it up to the user. The users created setuid shells in many different locations, some as "dot files" to be hidden from a normal `ls` listing, and some just setuid existing shells. In all cases, as none of the users made this rogue modification in the same manner, ISE-T was able to detect the rogue changes as a difference that existed between all the environments.

6 Related Work

Two-person control mechanisms are known to provide high levels of assurance [22]. Multiple examples exist with regard to nuclear weapons. For instance, to launch a nuclear weapon, two operators must separately confirm that launch orders are valid and must turn their launch keys together to launch the missiles. In fact, every sensitive action concerning nuclear weapons must be performed by two people with the same training and authority [3, Chapter 2]. The same notion is applied in many financial settings: banks will require two people to be involved in certain tasks, such as opening a safe-deposit box [25], and companies can require two people to sign a check [8] over a certain threshold. This makes it much more difficult for a single person to commit fraud.

However, as far as we know, this mechanism has never been applied directly to system administration. In the Compartmented Mode Workstation (CMW), the system administration job is split into roles, so that many traditional administration actions require more than one user's involvement [23]. These demarcation of roles were first pioneered in Multics at MIT [12]. Similarly, the Clark-Wilson model was designed to prevent unauthorized and improper modifications to a system to ensure its integrity [4]. All these systems simply divided the administrators' actions amongst different users who performed different actions. This differs fundamentally from the traditional notion of two-person control where both people do the same exact action.

More recently, many products have been created to help prevent and detect when accidental mistakes occur in a system. SudoSH [9] is able to provide a higher level of assurance during system administration as it records all keystrokes entered during a session and is able to replay the session. However, while sudosh can provide an audit log of what the administrator did, it does not provide the assurances provided by the two-person control model. Even if one were to audit the record or replay it, one is not guaranteed to get the same result. Although auditing this record can be useful for detecting accidental mistakes, it cannot detect malicious changes. For instance, a file fetched from the Internet can be modified. If the administrators can control which files are fetched, they can manipulate them before and after the sudosh session. ISE-T, on the other hand, does not care about the steps administrators take to accomplish a task, only the end result as it appears on the file system.

Part of the reason accidental mistakes occur is that knowledge is not easily passed between the experienced and inexperienced system administrators. Although systems like administration diaries and wikis can help, they do not easily associate specific administration actions with specific problems. Trackle [6] attempts to solve this by combining an issue tracker with a logged console session. Issues can be annotated, edited and cross-referenced while the logged console session logs all actions taken and file changes and stores them with the issue, improving institutional memory. Although this can help prevent mistakes from entering the system due to enabling the less experienced system administrators from seeing the exact same steps a previous administrator took to fix a similar or equivalent issue, it does not prevent mistakes from entering and remaining in the system, nor does it prevent a malicious administrator from performing malicious changes.

ISE-T's notion of file system views was first explored in Plan 9 [17]. In Plan 9, it is a fundamental part of the system's operation. As Plan 9 does not view manipulating the file system view as a privileged operation, each process can craft the namespace view it or its children will see. A more restricted notion of file system views is described by Ioannidis [11]. There, its purpose is to overlay a different set of permissions on an existing file system.

Finally, a common way to make a system tolerant of administration faults is to leverage the semantic of file system versioning, as it enable you to rollback to a configuration file's previous state when an error was made. Operating systems such as Tops-20 [7] and VMS [15] include native operating system support for versioning as a standard feature of their file systems. These operating systems employ a copy-on-write semantic that involves versioning a file each time a process changes it. Other file systems, such as VersionFS [16], ElephantFS [19], and CVFS [21] have been created to provide better control of the file system versioning semantic.

7 Conclusions

ISE-T applies the two-person control model to system administration. In administration, the two-person control model requires two administrators to perform the same administration act with equivalent results in order for the administration changes to be allowed to affect the system that is being modified. ISE-T creates multiple paral-

lel environments for the administrators to perform their administration changes and then compares the results of the administration changes for equivalence. When the results are equivalent, there is a high assurance that system administration faults have not been introduced into the system, be they malicious or accidental in nature.

We have implemented an ISE-T Linux prototype that creates parallel administration environments where separate administrators can perform changes, while not having administration rights on the machine itself. Our results from a user study demonstrate that many common administration tasks will result in equivalence when performed by isolated administrators without any communication between them. This demonstrates that the two-person control model can be applied to system administration by simply analyzing the results of the file system changes that occur in the environments created for the two administrators.

Acknowledgements

Paul Anderson, Andrew Hume, our paper shepherds, and Matthew Barr provided many helpful comments on earlier drafts of this paper, especially in the area of configuration management. This work was supported in part by NSF grants CNS-0426623, CNS-0717544, and CNS-0914845.

References

[1] *US DOD Joint Publication 1-02, DOD Dictionary of Military and Associated Terms (as amended through 9 June 2004).*

[2] P. Anderson. *LCFG: A Practical Tool for System Configuration*. Usenix Association, 2008.

[3] A. B. Carter, J. D. Steinbruner, and C. A. Zraket, editors. *Managing Nuclear Operations*. The Brookings Institution, Washington, DC, 1987.

[4] D. D. Clark and D. R. Wilson. A Comparison of Commercial and Military Computer Security Policies. *IEEE Symposium on Security and Privacy*, 0:184, 1987.

[5] Commission for Review of FBI Security Programs, William Webster, chair. Webster Report: A Review of FBI Security Programs, Mar. 2002.

[6] D. S. Crosta, M. J. Singleton, and B. A. Kuperman. Fighting Institutional Memory Loss: The Trackle Integrated Issue and Solution Tracking System. In *Proceedings of the 20th Large Installation System Administration (LISA 2006) Conference*, pages 287–298, Washington, DC, Dec. 2006.

[7] Digital Equipment Corporation. Tops-20 user's guide, Jan. 1980.

[8] M. S. Elmaleh. Nonprofit fraud prevention. `http://www.understand-accounting.net/Nonprofitfraudprevention.html`, 2007.

[9] D. Hanks. Sudosh. `http://sourceforge.net/projects/sudosh/`.

[10] J. Heller. *Catch-22*. Simon and Schuster, 1961.

[11] S. Ioannidis, S. M. Bellovin, and J. Smith. Sub-operating Systems: A New Approach to Application Security. In *SIGOPS European Workshop*, Sept. 2002.

[12] P. Karger. Personal Communication, May 2009.

[13] P. A. Karger and R. R. Schell. MULTICS Security Evaluation: Vulnerability Analysis. Technical Report ESD-TR-74-193, Mitre Corp, Bedford, MA, June 1977.

[14] O. Laadan, R. Baratto, D. Phung, S. Potter, and J. Nieh. DejaView: A Personal Virtual Computer Recorder. In *Proceedings of the* 21^{th} *ACM Symposium on Operating Systems Principles (SOSP)*, Oct. 2007.

[15] K. McCoy. *VMS File System Internals*. Digital Press, 1990.

[16] K. Muniswamy-Reddy, C. P. Wright, A. Himmer, and E. Zadok. A Versatile and User-Oriented Versioning File System. In *Proceedings of the Third USENIX Conference on File and Storage Technologies (FAST 2004)*, pages 115–128, San Francisco, CA, Mar./Apr. 2004. USENIX Association.

[17] R. Pike, D. L. Presotto, K. Thompson, and H. Trickey. Plan 9 from Bell Labs. In *Proceedings of the Summer 1990 UKUUG Conference*, pages 1–9, London, UK, July 1990. UKUUG.

[18] D. Price and A. Tucker. Solaris Zones: Operating System Support for Consolidating Commercial Workloads. In *Proceedings of the 18th Large Installation System Administration Conference*, Nov. 2004.

[19] D. S. Santry, M. J. Feeley, N. C. Hutchinson, A. C. Veitch, R. W. Carton, and J. Ofir. Deciding When to Forget in the Elephant File System. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles (SOSP)*, Dec. 1999.

[20] S. Soltesz, H. Pötzl, M. e. Fiuczynski, A. Bavier, and L. Peterson. Container-Based Operating System Virtualization: A Scalable, High-Performance Alternative to Hypervisors. *SIGOPS Operating System Review*, 41(3):275–287, 2007.

[21] C. A. N. Soules, G. R. Goodson, J. D. Strunk, and G. R. Ganger. Metadata Efficiency in a Comprehensive Versioning File System. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies*, Mar. 2003.

[22] P. Stein and P. Feaver. *Assuring Control of Nuclear Weapons*. University Press of America, 1987.

[23] J. S. Tolliver. Compartmented Mode Worksation (CMW) Comparisons. In *Proceedings of the 17th DOE Computer Security Group Training Conference*, Milwaukee, Wi, May 1995.

[24] VMware, Inc. `http://www.vmware.com`.

[25] Wilshire State Bank. Safe deposit boxes. `https://www.wilshirebank.com/public/additional_safedeposit.asp`, 2008.

[26] D. Wise. *Spy: The Inside Story of how the FBI's Robert Hanssen Betrayed America*. Random House, 2002.

[27] C. P. Wright, J. Dave, P. Gupta, H. Krishnan, D. P. Quigley, E. Zadok, and M. N. Zubair. Versatility and Unix Semantics in Namespace Unification. *ACM Transactions on Storage*, 2(1):1–32, Feb. 2006.

The Water Fountain vs. the Fire Hose: An Examination and Comparison of Two Large Enterprise Mail Service Migrations

Craig Stacey, Max Trefonides, Tim Kendall, Brian Finley
Argonne National Laboratory

Abstract

Mail administrators will inevitably face a situation where they will need to migrate their users from one server to another, not infrequently migrating to a different service altogether. In 2008, two divisions of Argonne National Laboratory found themselves needing to migrate their users from disparate divisional mail servers to a central, institutional Zimbra Collaboration Server. Each group approached the situation from a different direction, driven by different motivations, timelines, and external forces; each ultimately achieved its goal, one more smoothly than the other. The first migration was driven by a high sense of urgency resulting in a "fire hose" approach, an en masse move followed by a grand switchover; the second migration was a more measured "water fountain" approach, taking in lessons learned during the first migration. Examining the processes, decisions, and tools used in each conversion yields a roadmap of successes and pitfalls that should prove useful to any systems administrators facing a similar task, regardless of the timeline within which they must work.

1. Overview

Argonne National Laboratory is served by a central IT services division, the Computing and Information Systems (CIS) division. As well, many of the programmatic divisions have their own IT staffs of varying sizes. This paper focuses on the work of the IT support groups from two of those divisions, the Mathematics and Computer Science division (MCS) and the Materials Science Division (MSD).

CIS offers services, including e-mail, to any of the divisions at Argonne. Until 2008, this e-mail service was provided solely as Microsoft Exchange. In mid-2008, Argonne began offering a choice between Exchange and Zimbra Collaboration Suite.

Prior to this migration project, both MCS and MSD ran their own e-mail services rather than using the central mail services for varying reasons that will be detailed below. MCS and MSD each maintains its own IT support groups, providing a number of services besides e-mail. Diagrams detailing the flow of mail to these divisions both before and after this migration are included in the appendices.

MCS consists of nearly 200 researchers, programmers, students, and visitors, with another 250 external collaborators. The division is home to several hundred workstations, three large clusters, and other high-performance computing resources. Aside from managing this diverse group of resources, the group also provides standard IT services such as web, mail, data storage, backup, and networking services. Management of these resources and services is handled by a single IT organization, the MCS Systems team, comprising 10 individuals with varying skill sets and specialties, as well as anywhere from 1 to 4 undergraduate students each summer, depending on workload and availability of interesting projects.

MSD is the focal point for research in materials science at Argonne National Laboratory and consists of over 200 researchers, students and staff. The MSD IT Operations group supports this division, providing support for over 200 workstation and several small clusters. MSD IT Operations also provides standard IT services similar to those provided by the MCS Systems team. The IT Operations team comprises 3 full time employees and 2 part time co-op students.

2. Mathematics and Computer Science Division (The Fire Hose)

MCS ran its own mail services, with user mailboxes provided by Cyrus IMAP on an AIX server with 6 TB (available) of fibre channel attached storage, an installation that was set up in 1998. Approximately 500 user mailboxes were active at the time of this migration, with another 200 lying dormant as their owners forwarded their mail elsewhere, totaling approximately 450 GB of mailbox data.

In this section, the process is described from the perspective of the MCS Systems team, with a summary of the CIS perspective at the end.

2.1. MCS Decision Process

The existing mail system was showing its age. In 2006, MCS began the process of evaluating an upgrade path for mail services. Ultimately, we decided to go with the same approach we'd been using for the past 8 years; mailbox services provided by Cyrus, though instead using a Linux server since our AIX expertise was lacking.

While we try to avoid these situations, an extended period of limited funding, staffing changes, and an overcommitted IT staff resulted in systems and services getting replaced only when they broke or failed to meet the existing need. As this server was generally rock solid, it was often overlooked, and its replacement was not considered an urgent matter.

While testing and implementation plans were being put into place, we became aware of a growing desire within our user community for shared calendaring and other collaboration tools. We entered into a joint trial of the Zimbra Collaboration Server (ZCS) with CIS, with MCS's focus being the calendaring component.

Several months later, the mail upgrade project was stalled as a result of other emergencies; however, the Zimbra pilot was going well. Realizing that the ZCS service inherently provides mailbox services, we re-evaluated our mail server upgrade plan.

Our decision to continue to provide our own mail services was driven by a number of factors, but one of the main motivators was that we needed to be in control of the data and service. When something goes wrong, our users expect us to be able to fix it, and fix it quickly. The prospect of outsourcing our mail services and leaving our IT staff unable to directly support it did not appeal to either the IT staff or management. To an outsider, this may seem simply territorial, and there is certainly some truth to be found in that thinking. However, historically, the relationship between the organizations that would become MCS and CIS had its rough patches. MCS management preferred a nimble set of services focused solely on advancing its research, and many saw CIS as slow, bureaucratic, and controlling. Overcoming this prejudice was not an easy task, but this seemed an opportune time to try.

By virtue of the fact that the Zimbra experiment in CIS was in its pilot stage, coupled with the fact that MCS was its largest user base, MCS systems administrators were given administrative access to the service. After confirming that this access would be continued in the production-level service, we decided to make Zimbra the mailbox service for the division.

We note that the scope of this migration was limited specifically to user mailboxes. MCS was not going to cease providing mail services; we still ran Majordomo-based mailing lists (which would be converted to mailman lists later in the year), as well as trouble ticket systems for ourselves and other groups, and virtual domain services. Therefore our solution needed to be able to support our remaining the primary Mail Exchanger for the domains we controlled, sending user-bound mail to the central service. This Zimbra solution fit the bill nicely.

Work on the conversion began in earnest as our mail server was continuing to show its age. For instance, the release of Mac OS 10.5 brought with it a new version of Mail, which many of our users use. This new version handled offline IMAP actions in a slightly different fashion from previous versions, and it was a way that seemed to cause our version of Cyrus IMAP to choke. This resulted in repeated error messages to the users and an ever-growing list of queued actions, as each failure caused a new copy of the offline action to be queued. As one can imagine, this situation got less and less bearable as time went on. Additionally, large mailings could bring the service to a crawl, and we were entering into a time of year when drafts of proposals would regularly be sent to large distribution lists. While it may not be the most efficient way of collaborating on a document, e-mailing Word and PowerPoint documents is certainly the most prevalent method among our users. Most significantly, as errors would occur and failures became increasingly frequent with the advancing age of the server, we felt we were increasingly in danger of losing mail.

2.2. MCS Migration Plan and Implementation

Problems with our existing server notwithstanding, we had what was fundamentally a simple problem – find a way to move messages from one IMAP server to another. We did a fair amount of research to locate the existing tools that could accomplish this move, since something of our own construction would likely be too much effort for what should fundamentally be a solved problem. Based on this web research, consensus in the community seemed to be that using imapsync [Lamiral] would be the most reliable method of accomplishing this migration. Likewise, Zimbra's own recommendations in migrating to a new server recommended this

path of action [Zimbra]. In using this tool, however, we had to consider the limitations of our setup:

- The old mail server (cliff) was being pushed to its limits already; therefore our migration could not be too aggressive on the server.

- Because imapsync uses the IMAP protocol, it requires us to know the users' passwords on both systems. While we could set their passwords on the new Zimbra server, since they were not yet using it, we could not know their passwords on the existing IMAP server, as it was NIS-bound and using their regular workstation passwords.

We circumvented the first problem by limiting ourselves to two concurrent syncs – testing indicated this was an acceptable load. The password problem was more complex, but our situation allowed us to employ a creative workaround. Because our mail server was NIS-bound, we attempted to use a local */etc/passwd* entry on the machine, allowing us to login with our system password, and allowing the users to login with their NIS passwords as a fallback. Alas, this did not work on our version of AIX, but it did give us the idea that solved the problem. We could create entries for a "ghost" user on the mail server with the same UID and path as the real user that used our system password. At the same time, our script would modify the flat *mailboxes* file that Cyrus used to map mailboxes.

After testing confirmed our scripts were doing the right things, we ran the migration process day and night over a period of weeks. Various pitfalls were encountered along the way because of a number of situations our testing did not predict, such as disks filling up, networks going down, and previously undetected corruption in mailboxes.

Throughout the entire migration process, this corruption in mailboxes posed a significant challenge, as there was no predictable method to discover the corruption until we tried traversing the mailbox structure and reading individual messages; indeed, the messages appeared to be normal in any index of a mailbox, and the problems appeared only when the messages would be read. The most common corruption seemed to be in the oldest mailboxes; indeed, some employees had mail archives dating back into the late 1990s. While this in and of itself should not have caused a problem, the errors seemed to be caused by the varied (and no doubt interesting) lives these messages led. Some originated on the precursor to the mail system we were replacing (Sun OS 4.1.4's Mail with Sendmail and qpopper). These messages were stored in a monolithic mbox file, POPped off the server by the user into their e-mail client of choice, then later reimported into the Cyrus server via IMAP. Our most plausible theory is that changes in header format and attachment handling is what caused Cyrus to fail on loading these messages, as almost all of the corrupt messages were messages from Exchange users with attachments. These messages would have to be removed from the inbox by hand. Happily, while the IMAP clients seemed to fail on transferring or reading the messages, they were generally capable of deleting the messages; only a small handful of messages required a file-level delete and mailbox reindex.

Ultimately, it became clear that the IMAP-only method of syncing was not going to solve this problem; it was simply too slow. Connections were timing out on large mailboxes, resulting in incomplete data syncs. Also, the problems on cliff were steadily getting worse, and it became clear we needed an aggressive schedule for the migration. We came to the realization that our beloved-yet-overworked "little engine that could" was on the verge of switching from "I think I can" to "I think I'm done." As such, the slow-and-steady approach was beginning to look like it was a bigger risk than charging forward. It was the end of February, and an organization-wide maintenance window was scheduled for the weekend of April 26 and 27. We chose this weekend for the migration as most services would be down already, and users would expect a loss of service over that weekend.

Our second pass at moving the data involved getting the raw mailbox data onto the new server via rsync, using Zimbra's command line tools on the server to import the data into user mailboxes, and then using imapsync to synchronize the flags on the mailboxes with their counterparts on the old server.

To ensure we did not bring the main Zimbra server's network to a crawl during our data sync, we synced to a development server with the intention of mounting the disk on the production server for the import. As is becoming evident to the reader, things rarely worked out the way we planned them, and this was no exception. The initial sync of the data took an excruciatingly long time, though we were hopeful the import into the production server would be a much quicker operation, since the slowness of cliff would be out of the equation. Alas, the nature of the SAN defeated us; it turned out both the rsync *and* the mailbox import were I/O bound,

and our development server's SAN was not as robust as the primary storage on the production server.

Our self-imposed April 26 deadline was fast approaching, and our progress was indicating we would probably finish the initial data sync the day before we were to switch over. The IMAP syncs were alphabetical by user name, and this estimate was based on progress throughout the alphabetical list of usernames. Astute readers should be able to predict the next pitfall we hit – our largest mailbox was one of the last ones alphabetically. On average, our user mailboxes were several hundred megabytes, with users rarely crossing into the gigabyte range. This particular user had a mailbox of over 20 GB. We realized we would have to handle this mailbox out of band if we were to meet our deadline, and we started syncing it and other large mailboxes concurrently, outside the automated process.

On the morning of April 26, it was evident that the sync would not be finished. Because the outage window had already been announced, we plowed ahead and attempted to finish the migration, figuring a day for the heavy work and all of Sunday to finish and tie up loose ends.

We turned off all incoming mail and started the final sync of user mailboxes with imapsync, which would capture any messages that arrived since the initial sync, along with setting the message flags for the users. This script ran throughout the day as we set up the new rules on our SMTP relays to direct mail to the correct servers. Because the old mail server would still be processing Majordomo mailing lists, we had to detect whether mail was for a user or mailing list and route it accordingly. By Saturday night, things seemed to be progressing well, and our spot checks on mailboxes looked okay, except message flags did not seem to be getting set correctly.

On Sunday, it became clear what was going wrong; the "ghost" user on the old server had full access to the user's mailboxes but did not share the message flags, and all messages were seen as "new". (We later surmised that even though there was a single copy of each Cyrus indexing file per mailbox, that file was storing a set of flags for each username that accessed the mailbox, as opposed to each UID; thus, all messages were "New" to the ghost user.) Because the mail system was now down, we uncoupled cliff from NIS and used only the local */etc/passwd* file, allowing us to use the user's real mailbox with our system password. This strategy solved the message flag issue, but we became aware of another issue. The script calling imapsync was supposed to be nondestructive; messages deleted on the old server should not be deleted on the new server. Because of a misreading of the configuration, however, this was not the case, and messages *were* being purged from the new mailboxes in some cases. In theory, such purging should not have mattered. However, perceptive readers will remember we flipped the switch on delivery of new messages on Saturday morning. Hence, any new mail delivered to the new mailbox was being deleted as soon as that user's sync was run.

Also, because of mailbox corruption on the old server, a handful of users had their mailboxes emptied on the new server. We needed to reconstruct these mailboxes from backups, or in some "friendly user" cases (i.e., fellow sysadmins), the users restored their own mailboxes from their local backups.

The migration was not yet complete, but users were getting anxious. By 7:00 Sunday evening, users whose accounts we had deemed to be fully migrated were allowed into their new mailboxes. Surprisingly, this did not go poorly. In fact, the feedback we received during this "early access" period helped us in later diagnoses. From what we were hearing, we could determine that in most cases, where the user's mailboxes were small, the migration was a success. However, for users with large or complexly organized mailboxes, it became apparent rather quickly that the migration was not complete. Entire years' worth of mail were missing from the mailboxes of some users who kept large archives.

By Monday morning, it was clear we had much more work to do to finish the migration. We announced to the users that we had reason to suspect the migrated mailboxes were not complete and that we would instead implement an approach whereby new mail continued arriving at the new mail server, and users would migrate their own mail via their mail clients, with help from the IT support staff where required.

This manual user-initiated sync took place over the next two months in a gradual process, with most users being completely migrated by mid-May. In part we were able to accomplish this migration by announcing that the old server would be shut down at the end of May. As someone wise beyond his years once said, "Announce the demise of the old [system] well in advance of really discontinuing it" [Evard94].

In the cases of users with large or deeply nested folder hierarchies, we engaged in a great deal of "handholding" to guide them through the process. Unfortunately, these users tended to be among the less technical

savvy in the division, and as such the workload in that handholding was significant. Also, as outlined below, some mailboxes could not be migrated at all without some server-side tweaks.

We emphasize that the MCS users were marvelously patient throughout this process. Indeed, a key in maintaining this level of patience was proper communication. As noted in Tom Limoncelli's AT&T Network migration, a high level of communication and status updates will make the users feel more a part of the process (and less a victim of it) [Limoncelli97].

2.3. MCS Pitfalls and Lessons Learned

With each approach we devised, the plan seemed foolproof on paper, and at each step of the way, something popped up proving us wrong. The list of things that went wrong reads like a proof of "Murphy's law."

The combination of imapsync and our aging mail server were incapable of moving the mailboxes. In fact, IMAP itself had great difficulty in handing some of the user mailboxes. Often, users would archive mail into folders they would never again look at. As these folders grew in the number of messages contained therein, some reached a size that would make it impossible to access them over an IMAP client; as the old mail server struggled to stat the files, the connection would time out. To get around this situation, we would manually break up the mailboxes into smaller folders, reindex the folder, and begin anew.

The rsync of the mailbox data was restarted numerous times because of failing disks, high CPU loads, and network outages. In some cases these syncs had been running uninterrupted for days before crashing. With each restart, we lost precious time as file systems were compared.

The misconfiguration of imapsync in our migration script was a significant pitfall. By using imapsync incorrectly and losing messages, we undid a significant portion of the work that was accomplished. Human error is going to happen in any venture driven by humans and can be easily compounded by late, stress-filled nights that follow long, stress-filled days. In short, a simple typo of a flag was a devastating blow to both our progress and morale. A second set of eyes on these scripts would have gone a long way toward solving this problem.

Numerous restarts in various parts of this project plagued us. In the period between January 3 and April 25, we started from "square one" five times after a previous plan of action proved unworkable. Instead of having 4 months to migrate, we effectively had 2 weeks. This time constraint ramped up our stress levels, knowing that delaying the move could only exacerbate the situation, living in fear of the old mail server falling over.

All of the work we did to move the data from the old server to the new server was ultimately abandoned. This was, perhaps, the hardest blow to our collective psyche. The "brass ring" throughout this process was our knowing we'd done all the heavy lifting for our users, and they'd not have to deal with the migration themselves. Instead, not only did we go through a tremendous effort, but it was for naught.

Because a significant source of angst in this process was the lack of documentation, we continue to ensure we do not run into this in the future. Much of the old system was simply undocumented, existing only in the head of the previous mail administrator – clearly not a sustainable method of operating. We have ramped up our efforts in documenting processes and configurations, and we've ensured that more administrators are involved in the operation and configuration of the servers, avoiding the single-point-of-knowledge problem we typically faced.

The biggest contributing factor to our problems with this migration was related to the age of the hardware, operating system, and software of our production mail server. Combined with poor documentation, this left us with an aging mail system that for years had generally performed well with little intervention, and nothing but fading institutional memory on how to repair or tweak it. And, as is the case with any stable rock in a dynamic ecosystem, it had acquired roots and tendrils embedded in it that we are to this day still trying to disengage.

As noted in Section 2.1, the root cause of the age of this system was its generally working as expected during a period of time where only "squeaky wheels" got the oil. Economizing on hardware by holding off upgrades can often seem prudent, and sometimes unavoidable, but it almost certainly leads to an inflated TCO in the longer run. Tallying the amount of work hours involved in extricating a long used and encrusted system from a reasonably complex environment would be an interesting exercise. Following a long-term plan for regular retirement and refreshing of hardware would have gone a long way toward mitigating much of our problems.

A technical factor in this process was the Cyrus IMAP mailbox database. This monolithic flat text, single-file database used by the version of Cyrus IMAP that we were running proved to both hamper and help our migration. We were hampered because the file was fragile, had a rigid format dependent on tabs, spaces, and sorting (requiring a different sorting than provided by AIX's sort command), and was prone to corrupt the mail stream when things went wrong. It helped because we had an easily scriptable way to insert the systems users in order to be able to get access to the users' mailboxes, by ensuring the "ghost user" was either the first or last alphabetically (i.e. "aaaaaaaa" and "zzzzzzzz").

In Section 2, we mentioned that we had 500 active accounts and an additional 200 that were later determined to be dormant, resulting in our moving 40% more users than we needed to. We gave thought to indentifying the unused mailboxes prior to migration, so as to avoid the work of moving users who no longer existed. A small amount of effort was put to this task, but we soon called it off as we discovered most of these users had very small mailboxes, and weeding them out from the process would be more work than simply migrating everyone wholesale. With a slower approach, it's more likely we would have taken the time to cull these unused accounts prior to a move – it was largely a decision based on the time left and the level of effort available.

We point out that, over time, our account and resource expiration policies have been disabling and deleting these mailboxes, and almost all have been removed with little work on our behalf.

The next time we have to perform a migration of mailboxes, we'll be far more likely to employ the process we ended up using after all other plans failed. We would choose a cutover date when all new mail will be delivered to the new server, and allow users to migrate their own mail with help from IT support before an announced deadline wherein the old mail server would be shut off.

While we certainly engaged in testing, we failed to properly identify the edge cases. In some cases we chose what we expected to be difficult mailboxes on which to audition new migration methods, yet we had a knack for choosing examples that, while certainly large and well aged, were problem free. A better sampling for our testing would have gone a long way to identifying many of the pitfalls in advance of our migration deadline.

When coming up with our migration plan, CIS recommended we employ a more staged rollout. We opted to go "all-in" as we did not feel we had the luxury of the time required to engage in such a migration. Of course, the irony of this situation is the mail server we were convinced was going to fall over at any moment stayed up through the manual migration process. In fact, it was finally retired in August of 2009.

It's also important to consider that our group tries to make things as seamless for our users as possible, and all of our research indicated we *would* be able to accomplish this migration with little to no user impact. Aside from updating their mail client configurations, the only change our users were supposed to notice was a faster and more reliable mail service. We have certainly learned that this was too lofty a goal in the given circumstances.

2.4. CIS Challenges and Participation in the MCS Migration

From the CIS perspective, Zimbra had been very successful as a pilot service, but we had no true experience running Zimbra as a production service, or with any significant data or user load. Going from a dozen gigabytes of mail to trying to appropriately scale the system to instantly take on roughly half a terabyte of mail data and 500 users was a cause of some concern, and a bit of a challenge.

Zimbra allows for separation of disk volumes for performance and cost reasons. CIS provisioned the production system with separate volumes for redo logs, primary mail store, and secondary mail store, among others. Mail flow into the system, including messages added via IMAP, first land in the redo logs, then the primary mail store. A weekly scheduled Zimbra HSM process then migrates old mail from the primary mail store to the larger secondary mail store on lower-performance, less expensive disk.

One unanticipated effect of the "fire hose" approach was the need to closely monitor volume consumption on these separate volumes; in particular the redo log and primary mail store volumes, neither of which was intended to be able to completely contain the amount of data being transferred during the MCS migration.

As the redo log volume filled up, it was necessary to manually invoke an incremental backup using Zimbra's self-backup facility. The Zimbra self-backup facility allows for atomic point in time restores, and does so by replaying appropriate bits from the redo-logs, which it

copies to a backup volume during incremental backups. As the primary mail store filled up, it was necessary to preemptively invoke the Zimbra HSM facility. Fortunately, message age persisted in the migrated mail, therefore allowing this process to work.

The HSM process, as the solution to the primary mail store filling up, was fairly easy to identify. It just made sense, we already understood how it worked, and had intended it for this purpose, just not on this schedule. On the other hand, we had no prior experience with the redo logs growing out of hand. Previously, the already scheduled daily incremental backups automatically handled them, so we had no prior need to pay them any notice – it just worked. This is a good example of the challenges of accurately modeling behavior of a system at scale in a small or simulated environment.

CIS wasn't too concerned about high load placed on the Zimbra server during the MCS migration, as they were the first production user base to migrate. In other words, if the migration caused performance issues, they would be affecting only themselves. This was a luxury that future groups making the migration would not be able to have.

3. Materials Science Division (The Water Fountain)

MSD ran an iPlanet mail server on a Sun server with approximately 120 mailboxes including service accounts. A majority of the mailboxes were active at the time of migration, as MSD had been doing some house cleaning to keep adequate free space. At the start of migration there were over 190 GB of mail.

In this section, the migration process is described from the perspective of the MSD IT Operation group.

3.1. MSD Decision Process

The current MSD IT operations staff had inherited an aging Sun e-mail server that was getting more costly to maintain. Maintenance contracts and the cost of adding additional storage were cost prohibitive because of the age of the server. Additionally, as the existing server had been installed and operated by administrators no longer with the division, there was a lack of expertise with this install.

MSD IT Operations was relatively new department to MSD, as IT support had been handled by an Argonne division that had been dissolved. Despite having a new IT staff, the division had inherited an aging IT infrastructure built and maintained by another group. Because of this older infrastructure MSD wanted to explore the possibility of using the CIS e-mail systems, yet we were apprehensive about relinquishing control. The division is accustomed to having its services run by a support group whose only responsibility is their own division. Bearing this in mind, we did give some consideration to bringing a new e-mail server online. But since we had so many other infrastructure problems to deal with, we felt the benefit far outweighed the consequences of migrating e-mail services to CIS. Additionally, using CIS e-mail gave us the advantage of using Argonne's central Active Directory authentication, as MSD users were tired of having several different authentication methods.

Since MSD had a large Mac OS user base, moving to CIS Exchange servers was not our first choice because of the various issues Mac OS users can have with connecting to Exchange. (Historically, the laboratory's Exchange server did not interact well with Entourage. This problem was solved after our migration was finished.) At this time we became aware of the CIS Zimbra pilot project and started a dialog with CIS and MCS regarding migrating to Zimbra. After MSD completed initial testing and conversations with both the Zimbra lead and the MCS lead, MSD joined MCS on the Zimbra pilot test. This was in the early spring of 2008, but unfortunately several other more urgent projects needed attention, delaying the start of planning of the migration until late July 2008. It was during this pilot test that MCS performed its migration. After the process was complete, the MSD administrators met with our MCS colleagues to discuss their process.

Since other commitments by IT staff had delayed work on the migration, we, too, started to feel a sense of urgency. We had two factors influencing our deadline; our maintenance contract on the Sun server was expiring in late 2008, and our SSL certificate would expire shortly after that. MSD did not want to incur the cost of renewing either of them, knowing the service was bound for decommissioning. Also, during a recent divisional review, there were many large e-mail attachments going back and forth among the users, resulting in one weekend where mail delivery came to a near standstill because of lack of storage space. Even after the review, it was a struggle to keep 10 GB free on the mail store.

Because divisional administrative support staff and senior management need to collaborate with others in the laboratory, a decision was made to migrate these

users to the central Exchange server. Otherwise, all MSD users were to be migrated to the Zimbra server.

3.2. The Plan and the Pitfalls

Once we decided to use Zimbra as our primary server, new employees received accounts on the Zimbra service. Initially this was limited to postdocs, since Argonne's Zimbra service was still technically in the pilot phase. With the installation of ZCS 5.0, it was officially moved to production status, and we started adding all new employees' mailboxes to the Zimbra server. This relieved some of the storage issues on the current MSD mail server, allowing MSD IT operations to work out the remainder of the migration planning without quite so much urgency.

MSD looked at using imapsync; but after meeting with MCS and discussing the problems they had with it, doing an all-at-once approach was ruled out. Among the several reasons not to use imapsync was the need to know the user's password; MSD would not have access to user's AD account password for the Zimbra e-mail accounts. Furthermore, from a general customer satisfaction perspective, doing one user at a time was far more appealing, as we could start with a few users and test the migration process, hammering out any issues. Other reasons included the experience of some of our IT staff with e-mail migrations from previous positions at other organizations that employed expensive third party tools to perform a behind the scenes migration. Based on this experience, MSD IT knew we would most likely end up touching every workstation anyway.

The process we settled on was a new feature available in Zimbra, the import component of the web interface. We used this tool because it off-loaded the migration from the client to the server. Thus, the migration process did not tie up the user's workstation during the move, which was especially beneficial when dealing with older machines or a large mailbox migration. Since the Zimbra Web Client (ZWC) allowed users to add and check external POP and IMAP accounts, we had the user log into the ZWC and add the user's old MSD account. This approach caused the Zimbra server to import all the user's mail completely as a server-side action, regardless of whether the user is logged in on the ZWC. During the mail import MSD changed the primary e-mail alias to point to the Zimbra server. Once the account had fully loaded in the web interface, we then moved and arranged the folders or contents of folders to the Zimbra account's mailbox tree to mirror the old MSD folder structure. Once completed, we deleted the old account from the ZWC and set up the user's e-mail client to access the new account.

During the migration MSD encountered some users that were off-site a vast majority of the time. To assist these users, MSD wrote up documentation on how to do their own migration. Additionally, some users preferred to do their own migration because it provided an opportunity to cleanup their e-mail.

After MSD started doing several migrations a day, the Zimbra server started to slow tremendously, affecting other division as well. Migrations were halted while the Zimbra team investigated. After finding the root cause was Zimbra's indexing of attachments, we decided to turn off this feature for the time being. With attachment indexing off, migrations were much faster, even with heavy e-mail users (5 GB+ mail boxes), and there was no impact on other users' experience with the system. This issue did not arise during the MCS migration, because no other users were interactively using the service during their migration, so the high machine load was not noticed.

Rather than simply moving alphabetically through the mailboxes, scheduling was done with some consideration to the user's mailbox size: we started with the smaller mailboxes to make sure the process was working. Once the process was established and server concerns were addressed, we based the schedule primarily on the user's convenience. We scheduled it in batches and tried to get as many done in one batch as possible.

With any migration like this, one must address setting user expectations accurately on access to the old data. MSD established a policy that a user's old e-mail account would remain accessible for 7 days after the migration but only through the web interface. After 7 days the password on the mail account was changed; after 30 days the account was deleted from the server. This policy was largely adhered to except in some instances requiring us to set up access to an old MSD mailbox because something was not migrated or we missed changing an e-mail alias.

Another hurdle was some users were having e-mail addressed to the fully qualified divisional e-mail address (user@division.anl.gov) instead of the main Argonne alias (user@anl.gov). In the setup that existed at the time, any mail sent to user@msd.anl.gov would be directed to Argonne's mail gateway, then handed off to our own mail server; and as long as that server was still in the migration process, that setup had to be maintained. Since migrated users simply had their @anl.gov

alias directed to their new Zimbra mailbox, they would not experience this problem, but these users who had distributed their internal MSD address needed their old e-mail account kept active longer while they alerted their senders and mailing lists. Other difficulties were the occasionally corrupted e-mail message on the old MSD mail server, as this would stop the Zimbra mail import. Once the corrupted e-mail message was deleted, the mail import would function as expected.

As a side-benefit of this migration, it allowed us to perform some account cleanup. MSD identified users who had retired but were still using their MSD mail account, as well as users who were forwarding their mail to outside services, a discouraged-but-within-policy practice.

We used the mail migration as an opportune time to update many systems to the latest versions of their e-mail client and web browser. For consistency purposes we used the Firefox web browser to perform the migration, but in this process we found some users still were using Firefox 1.0, a long-outdated version.

3.3. MSD Pitfalls and Lessons Leaned

MSD IT, with the insight gained by the MCS migration experience, was able to create a more controlled migration process. Our biggest hurdle was sticking to the plan: specifically, scheduling each user, keeping track of migrations, and following through with all users. Adhering to this last step proved problematic, because, once we had all but a few the users migrated, we let other issues take priority and the last of the migrations took a back seat. Unlike MCS, we thought our e-mail server running with a light load would last awhile. Despite our migration going generally smoother than MCS's, we were not immune to the assumption that would be proven quite demonstrably wrong.

Of this handful of accounts on the old server, most were service accounts, not used by any particular user. However, we did have two user accounts left. One was a former division director who proved difficult to schedule. Since he was moving to Exchange, his migration required more coordination with CIS, as their Exchange administrators would need to assist in the migration. We also had a user we thought had been migrated to another division's e-mail server because he had been transferred to that division, but who turned out to still be using our old server. At the time we were getting ready to start migrating these account, our aging (and now unsupported) Sun server crashed in spring 2009. Since another division was involved, we combined efforts to bring the server back up. But the server had experienced nearly catastrophic failure; the data drives were intact, but we had no access to them without spending considerable time and money.

Fortunately, the former division director had a local cached copy of most of his e-mail, and we were able to use this for the migration. Unfortunately, the other user accidently deleted his locally cached copy, and we were unable to recover all of his older e-mail. We are still exploring our options for recovery, but the server is still offline. We quickly recreated most of the service accounts, but we are still finding some as we continue to review mail logs.

We've learned to follow through on our tasks and see them to completion. Also, we will do a better job confirming that work we think is done *actually is done*. Moreover, documentation can be improved, and properly documenting which service accounts we've created and what they're used for will help us a great deal down the road.

3.4. CIS Challenges and Participation in the MSD Migration

From the CIS perspective, the MSD migration was much more straightforward than the MCS migration. MSD engaged CIS early in their process. Based on experience gained from the MCS migration, and new features available in Zimbra that MCS helped explore and test, CIS was able to work with MSD to create a migration plan that worked well for them and minimized the impact on the Zimbra service and on MSD by spreading the migration out over time.

Both MCS and MSD handled their own migrations, engaging CIS when necessary. After the initial planning phases, the MSD migration was much more hands off for CIS. The one exception was the attachment indexing issue mentioned above.

CIS imposes no limits on mailboxes in our Exchange and Zimbra services and allows individual messages as large as 100 MB. Some of the components of mail systems work well with smallish messages but exhibit strain when processing large messages. At the time of the MSD migration, the attachment indexing process was a multithreaded Java process that had issues handling large attachment sizes. The net result was a dramatic increase in load on the system, both for CPU and disk, resulting in the Zimbra server being so slow it was almost unusable. Upon identifying the offending process, we disabled attachment indexing via a simple check box in the Zimbra admin GUI, and migrations

were able to resume. We note, for Zimbra's sake, that there is a new facility that can be selected for attachment indexing that is proving to better handle large attachments, and is resulting in a consistently lower system load.

4. Conclusions

Hindsight is, of course, 20/20, and one can easily look at both migrations and conclude that it's obvious what to do and what to avoid. Of course, every situation is different, and a careful examination of what went wrong and *why* can often lead to insights on how to avoid similar pitfalls when one is pushed down a similar path. In this section, we look at what each of the divisions took away from the process, having seen the results from each other's migration.

4.1. MCS

In many ways, performing an e-mail migration like this is not unlike performing a number of other types of migrations in the IT world, whether it's physically moving a datacenter, or implementing a new network topology, or deploying a new authentication scheme. In other ways, however, they can be vastly different, and it's in recognizing these differences that we can make better choices. Outside influences, customer demands, and occasionally the laws of physics can get in the way of how we expect things to play out.

MCS would obviously opt for a more measured approach in future migrations. The plan employed by MSD holds great appeal; however, two important factors exist. First, this option was not available on the version of Zimbra the lab was running at the time of our migration. Second, testing on our old mail server indicated that this implementation would not have worked for much the same reason imapsync failed; an aging server combined with enormous mailboxes results in timeouts and dropped connections.

Instead, time permitting, a well-documented and user-driven migration would be our likely course of action when undertaking a migration of this size. As in the prior-cited Tenwen paper, we would build the new system separate from the old one, move the users' delivery to the new system, and help them move their old data to it on their own schedule, within the constraints of our ability to maintain and run that old system. After a well-publicized and finite period of time, we would decommission the old system [Evard94].

As a service organization, it is always an admirable goal to inconvenience one's users as little as possible, but there are situations, such as this, where it's simply not attainable. A side benefit of a user-driven migration is an increased likelihood that users will be more selective as to which data must be maintained – our users can be notoriously bad at pruning unneeded data, resulting in just the sort of bloat that led to some of the issues we faced.

However, time is not always flexible, and when faced with an immovable deadline, one sometimes has no alternative but to jump in with both feet and try to solve the problem to the best of one's ability. If one absolutely had to do a migration like this, our implementation plan could have worked with better parameters, though it would by no means be the preferred solution. Certainly, a longer outage window and fewer false starts would have helped, but significant user input would still be required because of the corruption in the data being moved. Aggressive scanning of the mailboxes using IMAP tools could have identified these problems well in advance and allowed us to repair or remove the troublesome data well in advance. Likewise, we could have front-loaded the heavy work by migrating the heaviest users first, rather than the easily scriptable alphabetical method. Indeed, when it became evident that certain users had disproportionately large mailboxes, we hand-started syncs on their mailboxes outside the automated process.

We note that in no way were the pitfalls and encumbrances the fault of the targeted mail server software or the server itself. We believe we would have faced these challenges regardless of the chosen path, largely because of the age of the existing mail server, and its inability to handle the volume of mail we were moving.

4.2. MSD

MSD's biggest issue was with actually completing the project. This left us with several loose ends we needed to deal with in crisis mode when the Sun server crashed, as opposed to a controlled shutdown of the old server.

The server crash notwithstanding, MSD would definitely use the same basic method again if faced with another similar migration, albeit with better follow-through. This user-centric migration allowed a lot of buy-in from the most important IT customer – the end user. It reduced the potential lost productivity of the scientist if a one-shot migration had been done. It was labor intensive for MSD IT Operations, but the benefit

of reaching out to the user on an individual basis reduced call volume and follow-up issues. Also, we were able to resolve most issues in a timely manner, instead of trying to deal with several dozen users at once.

4.3. Avoiding Disaster

Many papers have been written describing IT moves, including the already cited [Evard94, Limoncelli97], as well as [Schimmel93, Cha98], dealing with moves and migrations both physical and virtual. Every move is different; each comes with its own pitfalls. Every time a group undertakes a project of such magnitude, there exists the opportunity to achieve both fantastic successes and extraordinary failures. The right steps taken beforehand can tip the scales more in favor of the former. Included in the appendices is the premigration checklist that we can now construct from our experiences, and would have dearly loved to have read prior to beginning the project.

Author Biographies

Craig Stacey is a full time computer geek, part time stand-up comic, aspiring photographer and writer, passionate beer enthusiast, and frequent wearer of pants. He is also the IT manager for the Mathematics and Computer Science Division at Argonne National Laboratory and longs to spend more time doing system administration and less time doing paperwork. His e-mail address is stace@mcs.anl.gov, and he is fond of monkeys and robots.

Adam Max Trefonides has been a UNIX Systems Administrator for many years. Prior to holding his current position as a senior systems administrator in the Mathematics and Computer Science Division at Argonne National Lab he was responsible for the team that, among many other duties, took care of the central e-mail systems at the University of Chicago, (in other words e-mail was his fault). Prior to working for the computers he was a cross-country trucker, carpenter, welder, sculptor and unemployment recipient. He maintains his trucker license for when the Internet fad ends. His e-mail address is maxadam@mcs.anl.gov.

Tim Kendall is a systems administrator and the primary Mac specialist in the Materials Science Division at Argonne National Laboratory. He loves Science Fiction of all types and was a professional photographer for 18 years before switching to IT. He helps run the Two Way Street Coffee House that has been in operation since 1970 presenting live folk music every Friday night. His e-mail address is tkendall@anl.gov.

Brian Elliott Finley is the deputy manager of Unix, storage, and operations for the Computing and Information Systems division at Argonne National Laboratory and is the lead on the Argonne Zimbra project. He holds a number of technical certifications and has created, maintained, or otherwise contributed to several open source software projects, including SystemImager and WiFi Radar. Mr. Finley lives in Naperville, IL, US with his wife, four children, one large dog, and a toad. He can be reached at finley@anl.gov.

Acknowledgments

This work was supported by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract DE-AC02-06CH11357.

References

[Cha98] Lloyd Cha et al., "What to Do When the Lease Expires: A Moving Experience," in Proceedings of the Twelfth Systems Administration Conference (LISA '98), pp. 168-174, Boston, MA, 1998

[Evard94] Rémy Evard, "Tenwen: The Re-engineering of a Computing Environment," in 1994 LISA Proceedings, pp. 37-46, San Diego, CA, 1994

[Lamiral] **imapsync**, Gilles Lamiral (developer), http://www.linux-france.org/prj/imapsync/

[Limoncelli97] Tom Limoncelli, "Creating a Network for Lucent Bell Labs Research South," in 11th Systems Administration Conference (LISA '97) Proceedings, pp. 123-140, San Diego, CA, 1997

[Schimmel93] John Schimmel, "A Case Study on Moves and Mergers", in Seventh System Administration Conference (LISA '93), pp. 93-98, Monteray, CA, 1993

[Zimbra] Zimbra Wiki, "Mail Migration instructions," http://wiki.zimbra.com/index.php?title=Mail_Migration

Appendix: Suggested Premigration checklist

As noted in Section 4.3, this is the checklist MCS should have used, constructed from the experiences gained from not using such a checklist.

Two months prior to migration

1. Inform users of the migration plan. Encourage data clean-up. Make clear and obvious the date the new service will begin.
2. Ensure user mailboxes are free of corruption. Aggressively scan mailboxes for errors using IMAP protocols. Instruct users on methods to test for problem mailboxes, including deleting problem messages.
3. Archive inactive mailboxes, and take them offline.
4. Compare list of active mailboxes with log files to identify users who are not logging in to check mail. Flag potentially inactive accounts, attempt to notify owners.
5. Identify exceptionally large mailboxes and work with owners to identify actual user needs and expectations – perhaps the mail client is configured to never empty the trash, for example.

One month prior to migration

6. Repeat items 1 through 5.
7. Go over potentially inactive account list from step 4, identify those actually inactive (eg, owner unreachable), and archive them.
8. Identify all accounts to be migrated, and create them on new server.
9. Ensure new account creation process is creating mailboxes on existing server and new server.
10. Hold training session with users demonstrating migration procedure.

One week prior to migration

11. Repeat items 1 through 5.
12. Ensure all accounts to be migrated are ready for service.
13. Hold another training session demonstrating migration procedure.
14. Ensure adequate availability for IT staff on migration day and the days that follow.
15. Post mail client configuration instructions so users can be ready for the switch. Adjust centrally managed mail client configurations.

One day prior to migration

16. Reiterate new service date very publicly. Post signs, and website announcements, send e-mails.
17. Ensure configuration instructions for mail clients are trivially available, trivially locatable, and correct.
18. Re-ensure IT staff availability.

Migration day

19. Buy lunch for the IT staff.
20. Implement migration plan.

Appendix: MCS Migration Scripts and Configuration Files

imapsyncbatch.sh - *used to launch imap sync sessions between cliff and Zimbra, this file lived on a third host named "owney" as cliff's SSL implementation was too old to open encrypted IMAP sessions to the Zimbra server. This is the version that contains the errant "-- delete2" that resulted in deletions from the Zimbra folders. stage1.mcs.anl.gov was the temporary hostname for the Zimbra mailboxes during migration.*

```
#!/bin/bash

USER1="zzzzzzzz"
USER2=$1@stage1.mcs.anl.gov
HOST1=cliff.mcs.anl.gov
HOST2=zimbra.anl.gov
DATE=`date "+%Y-%m-%d_%H:%M:%S"`
EXCLUDE="Trash|Viral"
SPLIT1=20
PASS1=/root/migration_scripts/cpass
PASS2=/root/migration_scripts/zpass
logfile=/sandbox/zzzzzzzz/log/$1-imapsync.log
userlog=/sandbox/zzzzzzzz/log/imapsync.log


cd /sandbox/zzzzzzzz/tmp
echo `pwd` >> $logfile
## Begin IMAPSync
echo "" >> $logfile
echo "-----------------------------------" >> $logfile
echo "IMAPSync started for $1   $DATE" >> $logfile
echo "" >> $userlog
echo "-----------------------------------" >> $userlog
echo "IMAPSync started for $1   $DATE" >> $userlog
echo "Settings: Excluding: $EXCLUDE, $SPLIT1 messages per" >> $logfile
echo "" >> $logfile

               echo "Starting $USER2 at $DATE" >> $logfile
echo "" >> $logfile
               imapsync \
               --nosyncacls --syncinternaldates \
               --nofoldersizes \
               --split1 $SPLIT1 \
               --exclude $EXCLUDE \
               --host1 $HOST1 \
               --user1 $USER1 \
               --passfile1 $PASS1 \
               --port1 993 \
               --host2 $HOST2 \
               --user2 $USER2 \
               --passfile2 $PASS2 \
               --port2 993 \
               --ssl1 \
               --ssl2 \
               --noauthmd5 \
               --delete2 \
```

```
                --buffersize 8192000 \
                --regextrans2 's/^Journal$/Journal-old/i' \
                --regextrans2 's/^Briefcase$/Briefcase-old/i' \
                --regextrans2 's/^Calendar$/Calendar-old/i' \
                --regextrans2 's/^Contacts$/Contacts-old/i' \
                --regextrans2 's/^Notes$/Notes-old/i' \
                >> $logfile
                echo "$DATE Finished $USER2" >> $logfile
                echo "" >> $logfile
# need some sanity checks here?

echo "" >> $logfile
echo "IMAPSync Finished for $1  $DATE" >> $logfile
echo "-----------------------------------" >> $logfile
echo "" >> $userlog
echo "-----------------------------------" >> $userlog
echo "IMAPSync Finished for $1  $DATE" >> $userlog
```

linker-forward.sh - *used to create /var/imap/mailboxes file on cliff with ghost users. This version traverses the alphabet from a to z, linking the user being synced with the ghost user "aaaaaaaa." The script needed to maintain the sorting and whitespaces contained within the existing file. As noted at the bottom, this script directly calls the above "imapsyncbatch.sh" on owney via an SSH session. The end of that SSH session allows this script to increment to the next user. A similar script, linker-reverse.sh, performed a similar job, albeit from z to a, linking the user being synced to the "zzzzzzzz" ghost user.*

```
#!/bin/ksh -x

## /root/migration_scripts/linker-forward.sh
## created by maxadam@mcs.anl.gov 3/2008
## modified by stace@mcs.anl.gov 4/2008
## with input from many quarters
##
## This script prepares cliff for migrating a user to zimbra.
## It is designed to work in tandem with linker-reverse.sh,
## to add parallelprocessing.
## What it does:
## Generates the userlist
## Moves a link to a commented version of /etc/inetd.conf in
## place and refreshes imapd in order to halt any new imap
## connections.
## Cleans the aaaaaaaa user out of the /var/imap/mailboxes file
## and copies the file to a working copy
## Creates the symlink for the aaaaaaaa user that points to the
## mail directory
## Backs up the mailboxes file, appending the current username
## Copies the modified mailboxes file into place
## Re-enables imap
## Runs imapsyncbatch on owney with $user as the single argument
## over ssh

log=/var/log/linker-forward.log
lock=/root/migration_scripts/locked
if [ ! -f $log ]; then
```

```
        touch $log
fi
 for i in `grep user /var/imap/mailboxes | awk '{print $1}' | awk -F . '{print $2}'| sort -u |
egrep -v ^aaaaaaaa | egrep -v ^zzzzzzzz` ; do
  while [ -f $lock ]; do
    sleep 20
  done
  touch $lock
  inetdpid=`ps -ef | grep '[i]netd' | awk '{ print $2 }'`
  echo "`date "+%Y-%h-%d@%H:%M:%S"`     Linking mailboxes for user ${i} to zzzzzzzz" >> $log
  if  [ ! -f "/etc/inetd.conf.off" ] 2>&1 >> $log; then
        echo "/etc/inetd.conf.off does not exist or is not an ordinary file! exiting." >> $log
        exit 1
  elif  [ ! -f "/etc/inetd.conf.on" ] 2>&1 >> $log; then
        echo "/etc/inetd.conf.on does not exist or is not an ordinary file! exiting." >> $log
        exit 1
  elif [ ! -L "/etc/inetd.conf" ] 2>&1 >> $log; then
        echo "/etc/inetd.conf is not a symlink or does not exist! Exiting." >> $log
        exit 2
  else echo "`date "+%Y-%h-%d@%H:%M:%S"`        Halting imapd" >> $log
        rm /etc/inetd.conf
        ln -sf  /etc/inetd.conf.off /etc/inetd.conf
        kill -HUP $inetdpid
        echo "`date "+%Y-%h-%d@%H:%M:%S"`       imapd halted" >> $log
        cp /var/imap/mailboxes /var/imap/mailboxes.backup-forward
  fi
  echo "`date "+%Y-%h-%d@%H:%M:%S"`       Making links for ${i}" >> $log
  egrep -v ^user.zzzzzzzz /var/imap/mailboxes  > /var/imap/mailboxes-f.${i}
  egrep "default        ${i}    " /var/imap/mailboxes | \
       sed s/^user.${i}/user.zzzzzzzz/ | \
       sed s/"default   ${i}    "/"default      zzzzzzzz        "/ >> /var/imap/mailboxes-f.${i}
  if [ ! -s /var/imap/mailboxes-f.${i} ] ; then
        echo "`date "+%Y-%h-%d@%H:%M:%S"`        Abort, empty mailboxes file" >> $log
        rm /etc/inetd.conf
        ln -sf /etc/inetd.conf.on /etc/inetd.conf
        kill -HUP $inetdpid
        exit 3
  fi
  rm -f /var/spool/imap/user/zzzzzzzz
  ln -sf /var/spool/imap/user/${i} \
         /var/spool/imap/user/zzzzzzzz
  if ! /bin/ls -l /var/spool/imap/user/zzzzzzzz | grep ${i} 2>&1 >> $log ; then
       echo "`date "+%Y-%h-%d@%H:%M:%S"`        Abort, link bad" >> $log
        rm /etc/inetd.conf
        ln -sf /etc/inetd.conf.on /etc/inetd.conf
        kill -HUP $inetdpid
       exit 4
  fi
  echo "`date "+%Y-%h-%d@%H:%M:%S"`       Links made" >> $log
  echo "`date "+%Y-%h-%d@%H:%M:%S"`       Copying mailboxes-f.${i} to mailboxes" >> $log
  if [ -s /var/imap/mailboxes-f.${i} ] ; then
         cp /var/imap/mailboxes-f.${i} /var/imap/mailboxes
  else
        echo "`date "+%Y-%h-%d@%H:%M:%S"`        Abort, empty mailboxes file" >> $log
```

```
        rm /etc/inetd.conf
        ln -sf /etc/inetd.conf.on /etc/inetd.conf
        kill -HUP $inetdpid
        exit 5
  fi
  echo "`date "+%Y-%h-%d@%H:%M:%S"`       Attempting to restart imapd" >> $log
  if  [ ! -f "/etc/inetd.conf.off" ] 2>&1 >> $log; then
        echo "/etc/inetd.conf.off does not exist or is not an ordinary file! exiting." >> $log
        exit 1
  elif  [ ! -f "/etc/inetd.conf.on" ] 2>&1 >> $log; then
        echo "/etc/inetd.conf.on does not exist or is not an ordinary file! exiting." >> $log
        exit 1
  elif [ ! -L "/etc/inetd.conf" ] 2>&1 >> $log; then
        echo "/etc/inetd.conf is not a symlink or does not exist! Exiting." >> $log
        exit 2
  else echo "`date "+%Y-%h-%d@%H:%M:%S"`       Restarting imapd" >> $log
  rm /etc/inetd.conf
  ln -sf /etc/inetd.conf.on /etc/inetd.conf
        kill -HUP $inetdpid
  echo "`date "+%Y-%h-%d@%H:%M:%S"`       imapd restarted" >> $log
  fi
  sleep 1
  echo "`date "+%Y-%h-%d@%H:%M:%S"`       Starting imapsyncbatch for ${i} on owney" >> $log
  rm $lock
  ssh -t zzzzzzzz@owney.mcs.anl.gov /root/migration_scripts/imapsyncbatch.sh ${i}
done
```

/var/imap/mailboxes snippet - *head and tail of the /var/imap/mailboxes generated by the scripts above. Recall that, at the filesystem level, the ghost users' spool directories would be symlinks to the actual users' directories.*

```
user.aaaaaaaa   default aaaaaaaa        lrswipcda
user.aaaaaaaa.Quarantine        default aaaaaaaa        lrswipcda
user.aaaaaaaa.SPAM      default aaaaaaaa        lrswipcda
user.aaaaaaaa.Viral     default aaaaaaaa        lrswipcda
user.aaaaaaaa.sent-mail default aaaaaaaa        lrswipcda
user.aammar     default aammar  lrswipcda
user.aammar.Drafts      default aammar  lrswipcda
[…]
user.zzhang     default zzhang  lrswipcda
user.zzhang.Drafts      default zzhang  lrswipcda
user.zzhang.Quarantine  default zzhang  lrswipcda
user.zzhang.SPAM        default zzhang  lrswipcda
user.zzhang.Trash       default zzhang  lrswipcda
user.zzhang.Viral       default zzhang  lrswipcda
user.zzhang.sent-mail   default zzhang  lrswipcda
user.zzzzzzzz   default zzzzzzzz        lrswipcda
user.zzzzzzzz.Quarantine        default zzzzzzzz        lrswipcda
user.zzzzzzzz.SPAM      default zzzzzzzz        lrswipcda
user.zzzzzzzz.Viral     default zzzzzzzz        lrswipcda
user.zzzzzzzz.sent-mail default zzzzzzzz        lrswipcda
```

Appendix: Mail Routing Diagrams

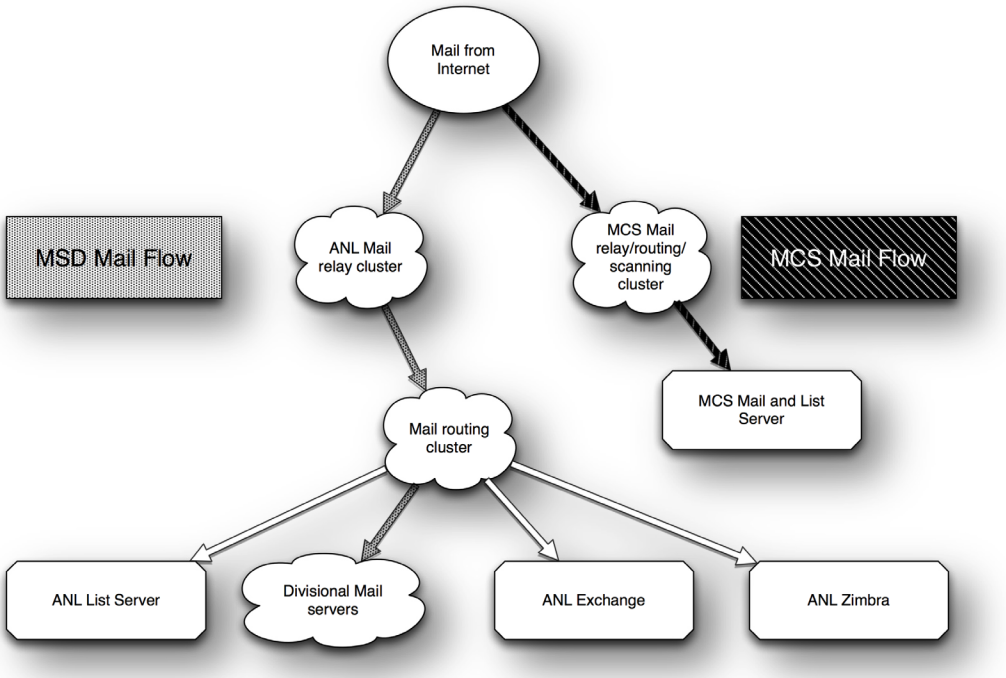


Figure 1 - Mail flow before migration project

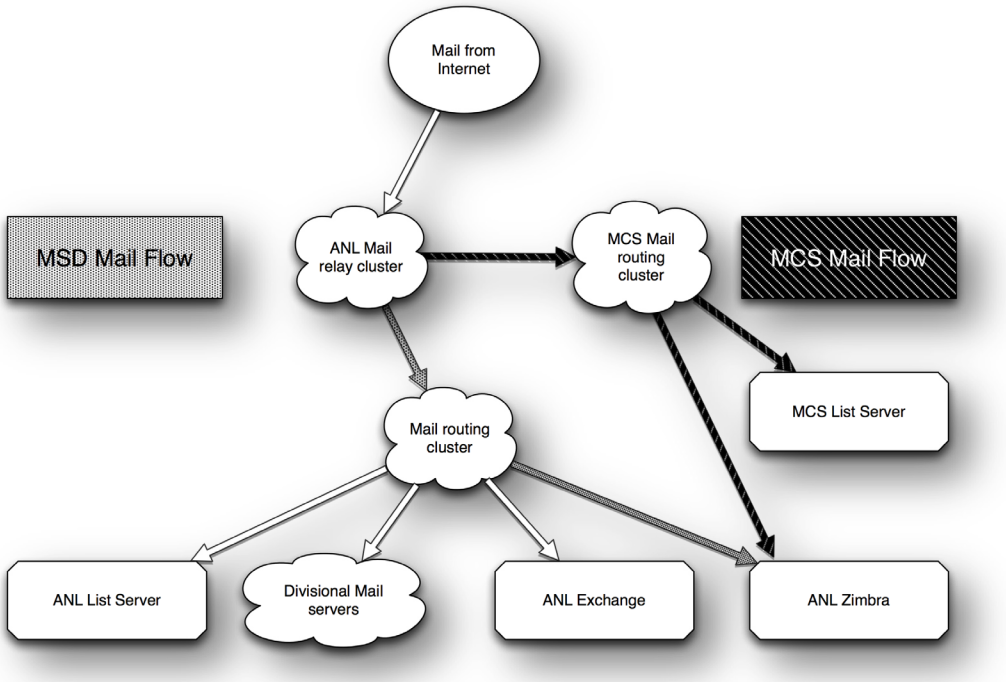


Figure 2 - Mail flow after migration project

Disclaimer – Non printing

Crossbow Virtual Wire: Network in a Box

Sunay Tripathi, Nicolas Droux, Kais Belgaied, Shrikrishna Khare
Solaris Kernel Networking, Sun Microsystems, Inc.

Abstract

Project Crossbow in OpenSolaris is introducing new abstractions that provide virtual network interface cards (VNICs) and virtual switches that can have dedicated hardware resources and bandwidth assigned to them. Multiple VNICs can be assigned to OpenSolaris zones to create virtual network machines (VNM) that provide higher level networking functionality like virtual routing, virtual load balancing, and so on. These components can be combined to build an arbitrarily complex virtual network called virtual wire (vWire) which can span one or more physical machines. vWires on the same physical network can be VLAN-separated and support dynamic migration of virtual machines, which is an essential feature for hosting and cloud operators.

vWires can be reduced to a set of rules and objects that can be easily modified or replicated. This ability is useful for abstracting out the application from the hardware and the network, and thus considerably facilitates management and hardware upgrade.

The administrative model is simple yet powerful. It allows administrators to validate their network architecture, do performance and bottleneck analysis, and debug existing problems in physical networks by replicating them in virtual form within a box.

Keywords: Virtualization, Virtual Switches, VMs, Xen, Zones, QoS, Networking, Crossbow, vWire, VNICs, VNM.

1 Introduction

In recent years, virtualization[2][3][7] has become mainstream. It allows the consolidation of multiple services or hosts on smaller number of hardware nodes to gain significant savings in terms of power consumption, management overhead, and data-center cabling. Virtualization also provides the flexibility to quickly repartition computing resources and redeploy applications based on resource utilization and hardware availability. Recently these concepts have enabled cloud computing[6] to emerge as a new paradigm for the deployment of distributed applications in hosted data-centers.

The benefits of virtualization is not only in consolidation and capacity management. With virtualization, the operating environment can be abstracted[14][18] and decoupled from the underlying hardware and physical network topology. Such abstraction allows for easier deployment, management, and hardware upgrades. As such focus has shifted towards multiple forms of network virtualization that do not impose a performance penalty[23].

Project Crossbow in OpenSolaris offers high performance VNICs to meet the networking needs of a virtualized server that is sensitive to network latency and throughput. Crossbow leverages advances in the network interface cards (NICs) hardware by creating hardware based VNICs which offer significantly less performance penalties. The VNICs have configurable link speeds, dedicated CPUs, and can be assigned VLAN tags, priorities, and other data link properties. Crossbow also provides virtual switches to help build a fully virtualized layer-2 network.

The VNICs can be created over physical NICs, link aggregations for high availability, or pseudo NICs to allow the administrator to build virtual switches independently from any hardware. Networking functionality such as routing and packet filtering can be encapsulated in a virtual machine or zone with dedicated VNICs to form virtual network machines. These virtual network machines can be deployed on virtual networks to provide layer-2 and layer-3 networking services, replacing physical routers, firewalls, load balancers, and so on.

With all the virtualized components Crossbow provides, an administrator can build an arbitrarily complex virtualized network based on the application needs and decouple it from the underlying physical network. The

resulting virtual network is called virtual wire. The vWire can be abstracted as a set of rules such as bandwidth limits, and objects such as VNICs and virtual switches, that can be combined, modified, or duplicated with ease and instantiated on any hardware. Crossbow allows migrating not just the virtual machine but entire virtualized network.

The functionality provided by Crossbow is part of the core OpenSolaris implementation, and does not require add-on products or packages.

In this paper we describe the main components of the Crossbow architecture from the perspective of a system and network administrator. We will introduce the new system and networking entities that are used for virtualizing the networking resources and for controlling the QoS at various granularities. We describe these entities with an emphasis on the simplified administration model by showing how they can be used as independent features, or as building blocks for the creation of vWires. In the examples section, we explore how Crossbow basic components can be used to build fully functional virtualized networks and new ways to do QoS. System administrators can also use the vWire to create a Network in a box to do performance, functionality, and bottleneck analysis.

2 Issues In Existing Models

The current methods of network virtualization are based on VLANs that are typically configured on the switches. This model is not very flexible if a VLAN tag is assigned to a virtual machine and the virtual machine needs to be migrated due to resource utilization needs. An administrator needs to manually add the virtual machine's VLAN tag to the switch port corresponding to the target machine. Protocols such as GVRP[13] and MVRP[17] are available for doing this dynamically. However, these protocols are not supported on a large number of switches.

The sharing of the common bandwidth between virtual machines also becomes an issue[9], as the current generation of switches offers fairness only on a per port basis. If the same port is shared by multiple virtual machines, any one of those virtual machines can monopolize usage of the underlying physical NIC resources and bandwidth. Host-based fairness or policy based sharing solutions impose significant performance penalties and are really complex to administer. They typically involve the creation of classes, the selection of queuing models, jitters, bursts, traffic selectors, and so on, all of which require an advanced knowledge of queuing theory.

Virtual networks that are created by using the existing VLANs and QoS mechanisms are prone to errors in the event of configuration changes or workload changes. The connectivity and performance testing is based on home grown solutions and requires expensive hardware based traffic analyzers. Often, there are heavy performance penalties and non-repeatable performance that depends on interactions with other virtual machines of different virtual networks.

This document will show how Crossbow can move VLAN separation and enforcement into the host and allow virtual machines to migrate without requiring changes to the physical network topology or switches. It will also show how VNICs can be associated with a link speed, CPUs, and NIC resources to efficiently and conveniently provide fair sharing of physical NICs. VNICs and virtual switches can be combined to build virtual networks which can be observed and analyzed by using advanced operating system tools such as DTrace.

3 Crossbow Virtualization Components

This section discusses the various Crossbow components that enable full virtualization, from virtualizing hardware resources such as NICs to building scalable vWire and network in a box.

3.1 Virtual NICs

When a host is virtualized, the virtual environment must provide virtual machines (VMs) connectivity to the network. One approach would be to dedicate one NIC to each virtual machine. While assigning dedicated NICs ensures the isolation of each VM's traffic from one another, this approach defeats one of the main purposes of virtualization, which is to reduce cost from the sharing of hardware. A more efficient and flexible option is to virtualize the hardware NICs themselves so that they can be shared among multiple VMs.

Crossbow provides the concept of the VNICs. A VNIC is created on top of a physical NIC, and multiple VNICs can share the same physical NIC. Each VNIC has a MAC address and appears to the system as any other NIC on the system. That is, VNICs can be configured from the IP stack directly, or they can be assigned to virtual machines or zones.

Crossbow can also assign dedicated hardware resources to VNICs to form *hardware lanes*. Most modern NIC hardware implementations offer hardware classification capabilities[10][20][12] which allow traffic for different MAC addresses, VLANs, or more generic traffic flows to be directed to groups of hardware rings or DMA channels. The Crossbow technology leverages these hardware capabilities by redirecting traffic to multiple VNICs in the hardware itself. The redistribution of traffic reduces network network virtualization overhead and provides better isolation between multiple VNICs that share the same underlying NIC.

In Crossbow VNICs are implemented by the OpenSolaris network stack as a combination of the virtualized MAC layer and a pseudo VNIC driver. The virtualized MAC layer interfaces with network device drivers under it, and provides a client interface for use by the network stack, VNICs, and other layered software components. The MAC layer also implements the virtual switching capabilities that are described in Section 3.3. The VNIC driver is a pseudo driver and works closely with the MAC layer to expose pseudo devices that can be managed by the rest of the OS as a regular NIC.

For best performance, the MAC layer provides a pass-through data-path for VNICs. This pass-through allows packets to be sent and received by VNICs clients without going through a bump-in-the-stack, and thus minimize the performance cost of virtualization. To assess the performance impact of VNICs, we measured the bi-directional throughput on a testbed consisting of 5 clients firing packets at a single receiver (quad-core, 2.8GHz, Intel-based machine) through a 10 Gigabit Ethernet switch. The measured performance of a VNIC with dedicated hardware lanes was the same as the performance of the physical NIC with no virtualization[24].

A side-effect of that architecture is that it is not possible to directly create VNICs over VNICs, although VNICs can be created on top of other VNICs indirectly from different OS instances.

Crossbow VNICs have their own dedicated MAC addresses and as such, they behave just like any other physical NIC in the system. If assigned to a virtual machine or zone, the VNIC enables that virtual machine to be reachable just like any other node in the network.

There are multiple ways to assign a MAC address to a VNIC:

Factory MAC address: some modern NICs such as Sun's 10 Gigabit Ethernet adapter[20] come from the factory with multiple MAC addresses values allocated from the vendor's MAC address organizationally unique identifier (OUI). VNICs can be assigned one of these MAC addresses if they are provided by the underlying NIC.

Random MAC address: A random MAC address can be assigned to a VNIC. The administrator can either specify a fixed prefix or use the default prefix. Crossbow will randomly generate the least significant bits of the address. Note that after a random MAC address is associated with a VNIC, Crossbow makes that association persistent across reboots of the host OS. To avoid conflicts between randomly generated MAC addresses and those of physical NICs, the default prefix uses an IEEE OUI with the local bit set. There is currently no guarantee that a randomly generated MAC address does not conflict with other MAC addresses on the network. This functionality will be delivered as part of future work.

Administratively set MAC Address: If the administrator manages the set of MAC addresses of the virtual machines or zones, he/she can supply the complete MAC address value to be assigned to a VNIC.

VNICs are managed by `dladm(1M)`, which is the command used to manage data links on OpenSolaris. Section 4.1.1 describes in details VNIC administration with the `dladm(1M)` command. A VNIC appears to the rest of the system as a regular physical NIC. It can be managed by other existing built-in tools such as `ifconfig(1M)`, or by third-party management tools.

VNICs have their own statistics to allow real time and historical analysis of network traffic that traverse them. Section 4.3 describes VNIC statistics and their analysis.

Last but not least, the traffic going through VNICs can be observed by existing tools such as `snoop(1M)`. Capturing packets going through VNICs is similar to observing the traffic on a physical switch port. That is, for a particular VNIC, only the broadcast and multicast traffic for the VLAN IDs associated with the VNIC, as well as the unicast traffic for the VNIC MAC address, are visible for observation.

3.2 Configurable Link Speeds

Transport protocol implementations will attempt to use the bandwidth that is made available by the underlying NIC[4]. Similarly, multiple VNICs defined on top of the same underlying NIC share the bandwidth of that NIC. Each VNIC will attempt to use as much as it can from the link's bandwidth. Various undesirable behaviors can ensue from this situation:

- A transport or a service can be an active offender – Some transport protocols are more aggressive than others. For example a UDP sender will not throttle its transmission rate even if the receiver cannot keep up with the received traffic. On the other hand, protocols like TCP will slow the sender down if needed. Such differences in behavior can lead to a VNIC for UDP traffic consuming more of the underlying bandwidth than other VNICs that are used for TCP.

- A client virtual machine can be a passive target of an external attack – In a virtualized setup where a hardware node is used to host virtual machines of different customers, one or more of those customers can become a victim of a denial of service attack[15][16]. The virtual machine for one customer can end up using most of the link's capacity,

effectively diminishing the performance of all the virtual machines that share the same NIC.

- Some VMs may have different bandwidth needs than others – The bandwidth of a NIC should be partitioned between VNICs to satisfy the requirements of the VMs. In some instances customers could be charged a premium if a larger share of the bandwidth is allocated to them. An uncontrolled or even egalitarian sharing of the resources might not necessarily be the desired behavior.

With the `dladm(1M)` command, Crossbow allows the link speed of data links to be specified through link properties. Configuring the link speed is the equivalent of setting a maximum bandwidth limit on the data link. This property can be configured explicitly by the administrator, or it can be set from the host OS of a virtualized environment when the VNIC for a virtual machine is created, as shown in Section 4.2 below.

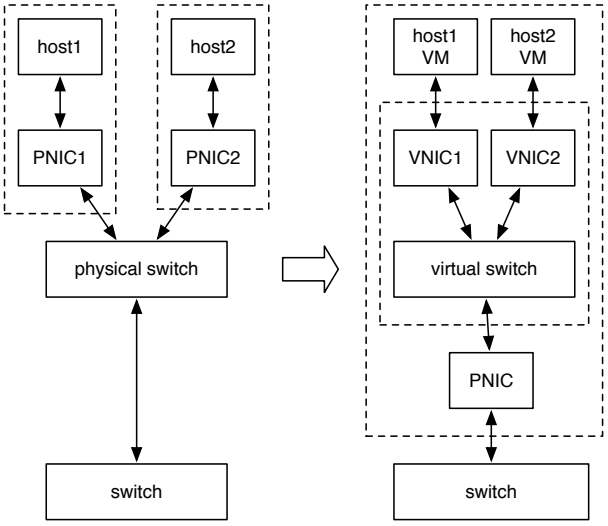


Figure 1: Mapping between physical and virtual switches

3.3 Virtual Switching

When multiple VNICs are created on top of a physical NIC, the MAC layer automatically creates a virtual switch on top of that NIC. All VNICs created on top of the physical NIC are connected to that virtual switch. The virtual switch provides the same semantics as a physical switch. Figure 1 shows the mapping between physical NICs and switches and their virtual equivalent in Crossbow. Note that multiple VNICs can be created on different physical NICs. In such cases, each physical NIC will be assigned its own virtual switch. Virtual switches are independent, and there are no data paths between them by default.

3.3.1 Outbound Packet Processing

When a packet is sent by a client of a VNIC, the virtual switch will classify the packet based on its destination MAC address. The following actions are taken depending on the result of that classification:

- If the destination MAC address matches the MAC address of another VNIC on top of the same physical NIC, the packet is passed directly to that VNIC without leaving the host.
- If the MAC address is a broadcast MAC address, a copy is sent to all VNICs created on top of the same physical NIC, and a copy is sent on the wire through the underlying NIC.
- If the MAC address is a multicast MAC address, a copy of the packet is sent to all VNICs which joined the corresponding MAC multicast group, and a copy is sent through the underlying NIC. The MAC virtual switch maintain a list of multicast membership for this purpose.
- If MAC destination is unknown, i.e. there is no entry for the MAC address in the layer-2 classification table of the virtual switch, the packet is passed down to the underlying physical NIC for transmission on the wire.

3.3.2 Inbound Packet Processing

Packets received off the wire are first classified by the NIC hardware according to the destination MAC address of the packet. If there is a match after hardware classification, the NIC hardware deposits the packet in one of the hardware rings associated with the MAC address. The MAC address and VNIC that are associated with that hardware ring is known to the host. Thus, when the host picks up the packet from that ring, it can deliver the packet to the correct VNIC network stack or virtual machine.

If the hardware classifier cannot find a dedicated hardware ring for the destination MAC address of the incoming packet, it deposits the packet in one of the dedicated hardware default receive rings. The MAC layer performs software classification on the packets received from these default rings to find the destination VNIC.

3.4 Etherstubs

We have seen in Section 3.3 that Crossbow creates a virtual switch between the VNICs sharing the same underlying physical NIC. As an alternative, VNICs can also

be created on top of *etherstubs* to create virtual switches which are independent of any hardware. Etherstubs are pseudo ethernet NICs and are managed by the system administrator. After an etherstub is created, it can be used instead of a physical NIC to create VNICs. The MAC layer will then perform virtual switching between the VNICs which share the same underlying etherstub.

Etherstubs and the MAC layer virtual switching allow users to create virtual switches which are independent from physical NICs. Whether the virtual switch is implicitly created over a link (physical NIC or an aggregation), or explicitly built by an etherstub, all VNICs sharing the same virtual switch are connected and can communicate with one another. Conversely, VNICs that are not members of the same virtual switch are isolated from each other. Figure 2 shows how virtual switching can be used between VNICs with both physical NICs and etherstubs.

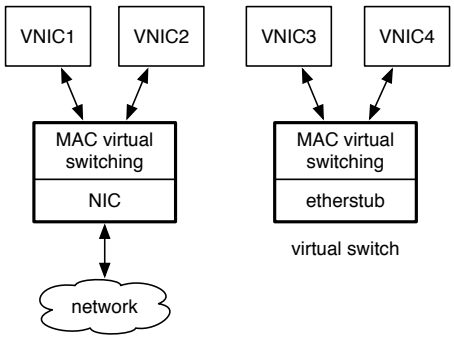


Figure 2: Virtual switching with physical NICs and etherstubs

Multiple etherstubs can be created to construct multiple virtual switches which can be combined to form flexible virtual networks. Section 5.2 shows an example of such an architecture.

3.5 VLANs

IEEE 802.1 VLANs can be used to build isolated virtual LANs sharing the same underlying physical layer-2 network infrastructure. Each VLAN is associated with a VLAN tag and defines its own broadcast domain. Hardware switches allow the traffic of different VLANs to be separated, and to associate switch ports with specific VLAN tags.

The Crossbow virtual switching is VLAN-aware and thus allows VLAN separation to extend to virtual switches and VNICs. VNICs can be associated with a *VLAN identifier*, or *VID*, which is used along with the MAC address to classify traffic to VNICs. As it is the case of physical switches, the Crossbow virtual switch also implements per-VLAN broadcast domains. In other words, tagged broadcast frames will be delivered only to the VNICs that match the VLAN tag. From the perspectives of efficiency and security, the Crossbow VLAN implementation provides two important features: it prevents the unnecessary duplication of frames and it ensures that no leakage of frames to the wrong VLAN is occurring.

Control of the VLAN handling is deliberately kept to the MAC layer of the host OS (or global zone when applicable). When a VNIC is used by a guest VM, the VM can only send and receive untagged traffic. The host's MAC layer inserts or strips the VLAN tag transparently. It also ensures that the VM does not attempt to send tagged packets. Thus, the VM cannot send packets on a VLAN to which it does not belong.

3.6 High Availability and VNICs

In order to provide highly available network connectivity, OpenSolaris supports availability at layer-2 and layer-3 by means of link aggregations and IPMP, respectively.

3.6.1 Layer-2: IEEE 802.3ad Link Aggregation

Link aggregations are formed by grouping multiple NICs in a single pseudo NIC. Multiple connections are spread through the NICs of the aggregation. Ports are taken out of the aggregation if they are misconfigured or fail unexpectedly. Failure detection is achieved by monitoring the link state of aggregated NICs or by exchanging Link Aggregation Control Protocol (LACP) control messages at regular intervals.

In OpenSolaris, link aggregations are managed by using `dladm(1M)` and implemented by a pseudo driver which registers with the system a pseudo NIC for each configured link aggregation. Each instance of the pseudo driver behaves like any other NIC on the system. As such, the pseudo driver allows VNICs to be created on top of link aggregations in the same manner that VNICs can be created on top of physical NICs or etherstubs. Figure 3 shows how two physical NICs can be aggregated, virtualized, and shared transparently by two guest domains.

The IEEE link aggregation standard assumes that an aggregation is built between two entities on the network. Typically these entities are switches and hosts. Unfortunately, this standard does not allow an aggregation to connect one host to multiple switches, which is a desirable configuration as a measure against possible switch failure. Some switch vendors have provided extensions called *switch stacking* that allow an aggregation to span multiple switches. These extensions are transparent to the peers that are connected to the switch stack.

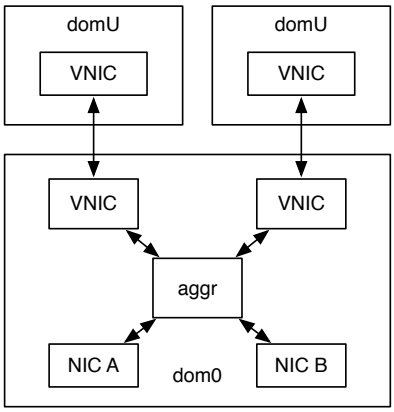


Figure 3: Using link aggregation to provide high-availability and increased throughput to VNICs

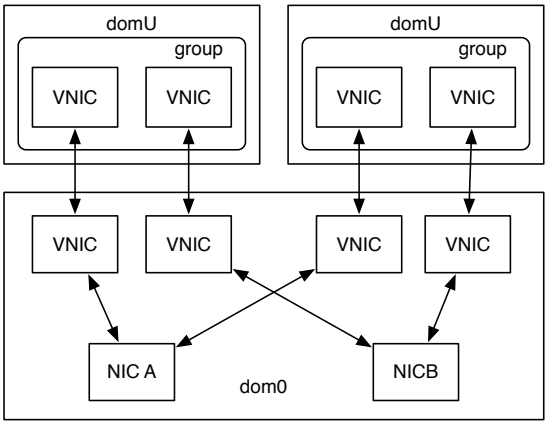


Figure 4: Using IP multipathing from virtual machines for high-availability

3.6.2 Layer-3: IP Multipathing

IP Multipathing, or IPMP[19], is a layer-3 high availability feature. It allows multiple IP interfaces to be grouped together, and provides load spreading and failover across members of the group. IPMP provides link-based detection failure, and probe-based detection failure.

Since IPMP is at layer-3 above NIC virtualization, VNICs cannot be created on IPMP groups and IPMP high availability cannot be provided transparently to virtual machines. Instead, VNICs can be created on each physical NIC, and VNICs can be grouped within virtual machines. Figure 4 shows how two NICs can be virtualized and grouped within virtual machines. IPMP groups are managed by using the `ifconfig(1M)` IP configuration tool.

Note that link aggregation and IPMP can be combined. For example, link aggregations can be used to group multiple NICs connected to the same switches, and IPMP can be used to group multiple link aggregations.

3.7 Virtual Network Machines

Virtual NICs and virtual switching constructs are the building blocks that allow more complex virtual networking topologies to be built within a host. The functionality needed to implement typical networking devices on a network, such as routers or firewalls, exists in modern operating systems like OpenSolaris. Networking devices can be therefore encapsulated within virtual machines or OpenSolaris zones.

An OpenSolaris zone is a lightweight virtualization architecture where the zone provides its own application environment that is isolated from other zones[21]. Each zone can be associated with a set of CPUs, data links such as VNIC, memory cap, and so on. Zones share the same kernel but each zone can have its own IP network stack. This feature avoids overheads that are typically associated with hypervisors. Because of their low overhead, small memory footprint, and specific functionality that does not require a full separate OS instance, zones are particularly suited to implement virtual network devices.

Virtual network machines refer to virtual machines or zones which are dedicated to implementing specific network functions. VNMs can be connected by assigning them VNICs and connecting these VNICs to virtual switches. Several types of network functions can be implemented, such as routers, firewalls, load balancers, and bridges. With Crossbow, essentially any layer-2 or layer-3 network can be virtualized within a single host.

3.8 Traffic Flows

Crossbow flows allow bandwidth limits, CPUs, and priorities to be associated with a subset of the network traffic that traverses a NIC, link aggregation, or VNIC. Flow attributes describe the traffic that is associated with the flows. Attributes consist of information such as IP addresses, well known port numbers, protocol types, and so on.

Crossbow flows span the whole network stack from the NIC hardware to sockets, and are associated with their own kernel threads and available hardware resources. Their specific associations make flows distinct from one another. Consequently, after hardware classification of incoming traffic is performed, traffic processing of flows can be scheduled independently from each other as well. With a setup that uses Crossbow, flows are better isolated, the task of classification is assumed by the hardware, and the network stack can control the arrival of traffic into the host on a per-flow basis.

Flows also maintain their own statistics to allow an administrator to track real-time statistics and usage history not only of individual data links as a whole but also of specific types of traffic the host receives or sends. Traffic flows are described in more detail in[25].

4 Ease of Management

Crossbow provides management tools that are easy to use to create VNICs, connect VNICs by using virtual switches to build vWires, and configure networking resources for these VNICs' dedicated use. In addition, statistics on traffic flows, both real time and historical, provide the administrator the ability to monitor traffic at a deeper granularity and thus better allocate networking resources. This section describes the Crossbow tools to perform these tasks.

4.1 Managing vWire

The vWire building blocks are managed through the `dladm(1M)` command, the OpenSolaris data-link management utility. This section shows how the `dladm(1M)` tool can be used to perform the following:

- Manage VNICs.
- Combine VNICs with etherstubs to build virtual networks.
- Combine VNICs with link aggregations to provide high availability and increased throughput to virtual machines and zones.

4.1.1 NIC Virtualization

As seen in Section 3.1, VNICs can be used to virtualize a data link. A VNIC is easily created with the `dladm(1M) create-vnic` subcommand. The following example shows the creation of a VNIC called `vnic100` on top of the physical NIC `e1000g4`.

```
# dladm create-vnic -l e1000g4 vnic100
```

In this case the administrator lets the system determine the MAC address to be associated with the VNIC. Users can choose any administratively meaningful name for the data links (NICs, VNICs, aggregations, etherstubs, and so on) as long as the name ends with a numeral. The `dladm(1M) show-vnic` subcommand can be used to display the VNIC configuration. For example:

```
# dladm show-vnic -o LINK,OVER
LINK         OVER
vnic100      e1000g4
# dladm show-vnic -o LINK,MACADDRESS
LINK         MACADDRESS
vnic100      2:8:20:36:ed:5
# dladm show-vnic -o LINK,OVER,MACADDRESS
LINK         OVER         MACADDRESS
vnic100      e1000g4      2:8:20:36:ed:5
```

The previous example shows how the `-o` option can be used to specify the fields to be displayed for each VNIC. If the `-o` option is omitted, then all attributes of the VNICs will be displayed.

VNIC attributes such as the specified MAC address to be associated with the VNIC can be specified by the user as additional options of `create-vnic`. The `dladm(1M) delete-vnic` subcommand can be used to delete previously created VNICs from the system. Of course, multiple VNICs can be created on top of the same physical NIC.

After a VNIC is created, it appears to the rest of the system as a regular data link and therefore can be managed in the same way as other NICs. It can be plumbed by the network stack directly as shown below, or assigned to a virtual machine as shown in Sections 4.2.1 and 4.2.2.

```
# ifconfig vnic100 plumb
# ifconfig vnic100 inet 10.20.20.1/24 up
# ifconfig vnic100
vnic100: flags=1000843<UP,BROADCAST,...
      inet 10.20.20.1 netmask ffffff00
      broadcast 10.20.20.255
      ether 2:8:20:36:ed:5
```

4.1.2 Etherstubs

Etherstubs are constructs that can be used to build virtual switches which are completely independent from physical NICs (see Section 3.4.) An etherstub can be used instead of a physical NIC to create VNICs. The VNICs sharing the same etherstub then appear to be connected through a virtual switch.

In the following example, an etherstub `vswitch0` is created, and then used to create three VNICs: `vnic0`, `vnic1`, and `vnic2`.

```
# dladm create-etherstub vswitch0
# dladm create-vnic -l vswitch0 vnic0
# dladm create-vnic -l vswitch0 vnic1
# dladm create-vnic -l vswitch0 vnic2
```

4.1.3 VLANs

Section 3.5 described how VLANs can be seamlessly integrated in the virtualization environment and used to create multiple virtual networks on the same underlying physical infrastructure. A VLAN can be easily associated with a VNIC during its creation.

```
# dladm create-vnic -l e1000g0 \
-v 200 vlan200vnic0
# dladm create-vnic -l e1000g0 \
-v 200 vlan200vnic1
# dladm create-vnic -l e1000g0 \
-v 300 vlan300vnic0

# dladm show-vnic -o LINK,MACADDRESS,VID
LINK         MACADDRESS           VID
vlan200vnic0 2:8:20:d5:38:7       200
vlan200vnic1 2:8:20:69:8f:ab      200
vlan300vnic0 2:8:20:3a:79:3a      300
```

As shown in the previous example, multiple VNICs can be created on top of the same physical NIC or etherstub with the same VID. In this case, the MAC layer virtual switching isolates these VLANs from each other, but will allow VNICs with the same VID to communicate together as if they were connected through a switch.

4.1.4 Link Aggregation

Link aggregations are also managed through the `dladm(1M)` utility. A link aggregation can be easily created as shown in the example below where an aggregation called `aggr0` consisting of two physical NICs, `e1000g2` and `e1000g3` is created.

```
# dladm create-aggr -l e1000g2 \
-l e1000g3 aggr0
```

The resulting `aggr0` is a regular data link on the system. It can be configured using `ifconfig(1M)`, or it can be used to create VNICs which are then assigned to zones or virtual machines. In the example below, two VNICs are created on top of `aggr0`:

```
# dladm create-vnic -l aggr0 vnic500
# dladm create-vnic -l aggr0 vnic501
```

4.1.5 Management Library

The `dladm(1M)` command is a thin CLI above the OpenSolaris data link management library libdladm. The bulk of the work is done by the library, while the command line tool implements the parsing and formatting needed. The libdladm management library is also used by other management tools, agents, and utilities.

4.1.6 Network Flows

Crossbow provides a new command `flowadm(1M)` to configure flows. As described in Section 3.8, flows can be used from vWire to control and measure bandwidth usage of finer grain traffic. The `flowadm(1M)` command takes as its arguments a data link name, traffic criteria, priority, and desired bandwidth. Traffic criteria can be specific protocols, protocol ports, or local or remote IP addresses.

For example, a flow to match all UDP traffic passing through NIC `ixgbe0` can be created as follows:

```
# flowadm add-flow -l ixgbe0\
-a transport=udp udp-flow
```

Each flow has associated properties specified by the `-p` option. These properties can be used to define the maximum bandwidth or priority for a flow. Properties of existing flows can be changed without impacting the flow's defined criteria. By default, `udp-flow` uses the bandwidth of the underlying NIC, which in the example is 10 Gb/s. To change the bandwidth of `udp-flow` to 3 Gb/s, issue the following command:

```
# flowadm set-flowprop -p maxbw=3G \
udp-flow
```

If no speed unit is specified, the `maxbw` property unit is assumed to be in megabits per second (Mb/s). Additionally, the `flowadm(1M) show-flow` and `show-flowprop` subcommands can be used to display flow configuration and properties respectively. Flows can be deleted using the `flowadm(1M) remove-flow` subcommand.

4.2 Resource partitioning and QoS

Configuring QoS policies often tends to be laborious. For example, a typical policy might be to limit TCP traffic to use a bandwidth of 1000 Mb/s. However, configuring such a policy by using IPQoS in Solaris 10[19] or tc[5] in Linux entails several complex steps such as defining queuing disciplines, classes, filter rules, and the relationships among all of them.

The subsections that follow use real life scenarios to illustrate how Crossbow vastly simplifies QoS configuration.

4.2.1 Zones

With Crossbow, limiting bandwidth for a zone is simple to perform. One just needs to create a virtual NIC with the desired bandwidth and assign it to the zone. For example, to limit the bandwidth of zone `zone1` to 100 Mb/s, first create a VNIC with the desired bandwidth:

```
# dladm create-vnic -p maxbw=100 \
-l e1000g0 vnic1
```

When the zone is created, it can be given `vnic1` as its network interface:

```
# zonecfg -z zone1
...
zonecfg:zone1> add net
zonecfg:zone1:net> set physical=vnic1
zonecfg:zone1:net> end
...
```

Any traffic sent and received `zone1` through `vnic1` will be limited to 100 Mb/s. The configuration steps are a one time exercise. The configuration will be persistent across the zone or the operating system reboot. Changing the bandwidth limit at a later time can be achieved by setting `maxbw` property of that VNIC to the new value. Thus, to change bandwidth of `zone1` to 200 Mb/s, use the following command syntax:

```
# dladm set-linkprop -p maxbw=200 vnic1
```

One can query the VNIC property `zone` to determine if the VNIC is assigned to any zone. Using the previous example, `zone` under the `VALUE` field indicates that vnic1 is a link that is being used by `zone1`.

```
# dladm show-linkprop -p zone vnic1
LINK         PROPERTY        PERM VALUE
vnic1        zone            rw   zone1
```

Plans are currently under consideration to configure zones' VNICs and their bandwidth limits directly by using `zonecfg(1M)`. Thus, VNICs with specific property values can be created automatically when the zones are booted.

4.2.2 Xen

When OpenSolaris is used as dom0 (host OS), Crossbow provides a simple mechanism to assign bandwidth limits to domUs (VM guests). The configuration process is similar to configuring bandwidth limits for zones. A VNIC is created with the desired bandwidth limit, and then supplied as an argument during domU creation. The domU could be running OpenSolaris, Solaris 10, Linux, Windows, or any other Xen supported guest. This process is independent of the choice of the domU. The procedure is explained in detail as follows:

When a Xen domU is created, Crossbow implicitly creates a VNIC and assigns it to the domU. To enforce a bandwidth limit for a domU, first, explicitly create a VNIC and assign it to domU during creation. Then, set the bandwidth limit for the Xen domU by setting the `maxbw` property of the VNIC.

For example, to limit the bandwidth of domU `guest1` to 300Mb/s, the VNIC with the given bandwidth is first created:

```
# dladm create-vnic -p maxbw=300 \
-l e1000g0 vnic1
```

Then, to assign the newly configured VNIC to the Xen domU as its network interface, include the following in the domU's `template.xml` configuration file. Use the `dladm(1M) show-vnic` subcommand to display the MAC address of `vnic1`.

```
<interface type='bridge'>
<source bridge='vnic1'/>
<mac address='vnic1's mac address/>
<script path='vif-dedicated'/>
</interface>
```

Finally, the domU is created as follows:

```
# virsh create template.xml
```

Any traffic sent and received by the guest domain through `vnic1` will be limited to 300 Mb/s. As with zones, the bandwidth can be changed at a later time by setting the `maxbw` property to the new value.

Plans are under consideration to configure bandwidth limit for Xen domUs by using Xen configuration tools such as `xm(1M)` and `virt-install(1M)`. For example, the `virsh-attach` interface command will take the maximum bandwidth as an optional argument. The specific bandwidth limit is then automatically applied to the implicitly created VNIC when the domain is booted.

When using Linux as dom0, bandwidth control on guests can be configured as follows:[1]

1. Associate a queuing discipline with a network interface (tc qdisc).
2. Define classes with the desired bandwidth within this queuing discipline (tc class).
3. Using the IP address of the guest OS's interface, define a rule to classify an outgoing packet into one of the defined classes (tc filter).

For example, the following set of commands issued from dom0, would set bandwidth limits of 200 Mb/s and 300 Mb/s for each one of the domU instances, and reserve the remaining 500 Mb/s for dom0' use[8].

```
# tc qdisc add dev peth0 \
root handle 1: htb default 99

# tc class add dev peth0 \
parent 1: classid 1:1 htb rate 1000mbps \
burst 15k

# tc class add dev peth0 parent 1:1 \
```

[1] At the time of writing this paper, the latest Fedora release that could host Xen guests was Fedora 8 (Fedora 9 and Fedora 10 cannot host Xen guests). It supports a vif parameter '`rate`'to control bandwidth limit. However, due to a bug (RedHat bug id 432411), we could not evaluate that feature.

```
classid 1:13 htb rate 200mbps burst 15k
# tc class add dev peth0 parent 1:1 \
classid 1:14 htb rate 300mbps burst 15k
# tc class add dev peth0 parent 1:1 \
classid 1:99 htb rate 500mbps burst 15k


# iptables -t mangle -A POSTROUTING \
-p tcp -s 192.168.1.103 -j CLASSIFY \
--set-class 1:13
# iptables -t mangle -A POSTROUTING \
-p tcp -s 192.168.1.104 -j CLASSIFY \
--set-class 1:14
# iptables -t mangle -A POSTROUTING \
-p tcp -s 192.168.1.111 -j CLASSIFY \
--set-class 1:21
```

Note that the previous approach does not work well when domUs obtain IP addresses by using DHCP. Moreover, domU users can circumvent the bandwidth limit enforcement by changing their IP address.

4.2.3 Traffic Flows

In the previous example, we restricted all traffic passing through a Xen domU to 300 Mb/s. Suppose that we further want to partition the available 300 Mb/s bandwidth as follows: 100 Mb/s for all TCP traffic and the remaining 200 Mb/s for all other traffic. Crossbow can achieve this configuration by using flows:

```
# flowadm add-flow -p maxbw=100 \
-a transport=tcp -l vnic1 tcp-flow1
```

The concept of flows is applicable to non-virtualized context as well. For example, a physical NIC can be specified instead of a VNIC. Thus, Crossbow provides a simple yet powerful way to administer bandwidth.

In contrast, configuring policies with `iproute(8)` and `tc(8)` on Linux typically involves several steps, For example:

```
# tc qdisc add dev eth4 handle ffff: \
ingress

# tc filter add dev eth4 parent ffff: \
protocol ip prio 20 \
u32 match ip protocol 6 0xff \
police rate 1Gbit buffer 1M drop \
flowid :1

# tc qdisc add dev eth4 root \
handle 1:0 cbq bandwidth 10Gbit \
avpkt 1000 cell 8

# tc class add dev eth4 parent 1:0 \
classid 1:1 cbq bandwidth 10Gbit \
rate 10Gbit prio 8 \
allot 1514 cell 8 maxburst 20 \
avpkt 1000 bounded

# tc class add dev eth4 parent 1:1 \
classid 1:3 cbq bandwidth 10Gbit \
rate 1Gbit weight 0.1Gbit prio 5 \
allot 1514 cell 8 maxburst 20 \
avpkt 1000

# tc class add dev eth4 parent 1:1
classid 1:4 cbq bandwidth 10Gbit \
rate 9Gbit weight 0.9Gbit prio 5 \
allot 1514 cell 8 maxburst 20 \
avpkt 1000

# tc qdisc add dev eth4 parent 1:3 \
handle 30: pfifo

# tc qdisc add dev eth4 parent 1:4 \
handle 40: pfifo

# tc filter add dev eth4 parent 1:0 \
protocol ip prio 1 u32 match ip \
protocol 6 0xff flowid 1:3
```

4.2.4 Flow Tradeoffs

The Crossbow design has traded off richness of flow attributes for simplicity and performance. Crossbow has departed from the traditional ways to specify QoS that consists of the following steps:

- Definition of classes of services
- Addition of rules similar to those of packet filtering
- Description of the packets that are assigned to each class

Instead, a flow is created by specifying its defining attributes that constitute as the common criteria that packets should match in order to belong to that flow. Resource controls policies, such as bandwidth constraints, priority and CPUs are viewed as mutable properties that can be allotted to flows at creation time and can be modified later.

Although flows can be created based on different attributes such as IP addresses, subnets, transport, DSCP marking, and port number, flows are defined based only on one attribute at a time, not on a combination of multiple attributes. Furthermore, only non overlapping flows are allowed to co-exist over a data link. Any attempt to create a flow that conflicts with an existing one fails. This apparent limitation provides the advantage of keeping the rule set that describes the flows inside the system unambiguous and order independent. A lookup for the flow

that matches a packet will always find the same flow, regardless of the presence of other flows or the order in which they were added.

4.3 Monitoring Network Statistics

Crossbow also provides a rich set of statistics for gaining better insight into the behavior of the system. This section describes the tools provided to observe these statistics, and concludes with an example scenario to illustrate how these tools can be combined with other commands to diagnose and resolve a performance issue.

4.3.1 dlstat(1M) and flowstat(1M)

Crossbow statistics are provided on a per flow or data link basis. They provide information such as the count of packets received by polling and by interrupts, hardware and software packet drops, distribution of load across hardware lanes and so on. These statistics help to identify performance bottlenecks.

The current interface provides counts over a certain interval. Future improvements will provide more sophisticated aggregate level statistics such as percentage of polled packets, minimum, maximum, and average queue lengths over a specified time interval, and so on.

Crossbow introduces `dlstat(1m)` to print dynamic traffic statistics for links. For example, the following command prints the aggregate statistics for `vnic1`:

```
# dlstat vnic1
LINK   IPKTS IBYTES OPKTS  OBYTES
vnic1   9.9M   2.3G  4.8M    0.3G
```

To observe traffic exchange at 5-second interval, use the following:

```
# dlstat -i 5 vnic1
LINK   IPKTS IBYTES OPKTS  OBYTES
vnic1   1.5M   0.3G  0.6M   46.9M
vnic1   2.2M   0.5G  1.1M   73.3M
  .      .      .     .      .
```

Apart from dynamic statistics, `dlstat(1M)` also supports off-line viewing and analysis of statistics. `acctadm(1m)` is used to enable logging network statistics to a specific log file. The `dlstat(1M) -u` sub-option can then operate on the log file to extract historical network statistics. For example, the following command will extract network statistics for `vnic1` from the specified time range from `logfile`.

```
# dlstat -u -f logfile \
-s D1,shh:smm:sss -e D1,ehh:emm:ess vnic1
```

The output, if generated using `-F gnuplot` option, could be directly fed to `gnuplot` to draw graphical usage information for `vnic1`.

To analyze detailed receiver side statistics such as poll and interrupt packet counts as well as hardware and software drops, do the following:

```
# dlstat -r
LINK     IBYTES INTRS POLLS HDRPS
e1000g0    2.1M 22.3K  78.0   0.0
ixgbe0    13.6G  0.8K 10.7M   0.0
vnic1     13.6G  0.8K 10.7M   0.0
```

To also analyze per hardware lane statistics, append the `-L` option to the previous command. For example, the following will show per hardware lane statistics for each hardware lane that belongs to `ixgbe0`.

```
# dlstat -r -L ixgbe0
LINK:LNE LTYP  USEDBY IBYTES INTRS POLLS
ixgbe0:0 slne  ixgbe0  13.6G  0.8K   0.0
ixgbe0:1 hlne  ixgbe0  13.1G  0.8K 10.2M
    .      .       .     .     .     .
    .      .       .     .     .     .
ixgbe0:7 hlne  ixgbe0  13.4G  0.8K 10.5M
```

While `dlstat(1M)` operates on data links, `flowsat(1M)` is used for querying network statistics for flows. For example, to display `tcp-flow`'s network traffic statistics, do the following:

```
# flowstat tcp-flow
FLOW     LINK  IBYTES OPKTS  OBYTES
tcp-flow vnic1   2.3G  4.8M    0.3G
```

Like `dlstat(1M)`, `flowstat(1M)` also supports logging network statistics by using the `-u` sub-option.

Both inbound and outbound traffic statistics are shown by `dlstat(1M)` and and `flowstat(1M)`. The bandwidth limits apply to the combined bidirectional traffic, which is the sum of incoming and outgoing packets over time. Although we can observe the statistics for each direction, we currently can't set a different limit on each.

4.3.2 Example: Diagnosing a Scalability Issue

Consider a multi-processor system under heavy network load that uses the NIC `ixgbe0` and whose receiver side network performance needs improvement. Suppose that the output of `dlstat -r -L` is satisfactory. That is, after listing per-hardware lane packet and byte counts as well as poll and interrupt counts, you observe that traffic is evenly distributed across hardware lanes and that 95% of packets are delivered by polling. You can then check CPU utilization as follows:

- `dlstat -r -F ixgbe0` gives the breakdown of which CPUs are currently being used to process packets received by `ixgbe0`.
- `dladm show-linkprop -p cpus ixgbe0` displays the list of CPUs associated with the data link.
- `mpstat(1M)` provides information about the utilization of each CPU that is associated with `ixgbe0`.

Suppose that the data indicates that all the CPUs that are currently assigned to `ixgbe0` for packet processing are fully utilized while other CPUs in the system are at an idle or near-idle state. To dedicate a new list of CPUs for `ixgbe0`'s use, the following command syntax is used:

```
# dladm set-linkprop \
-p cpus=<list of cpus> ixgbe0
```

5 Virtual Wire: Network in a Box

We have described so far the major components needed for achieving network virtualization using convenient and intuitive tools. We then showed how bandwidth and computing resources can be awarded and controlled at a fine granularity to data links and VNMs. We can now use the VNMs, VNICs, etherstubs, along with the virtual switching and resource control capabilities as the building blocks to construct fully functional vWires of arbitrarily complex topologies in a single or small set of machines. The three scenarios below are examples of vWires used for consolidation of subnet and enterprise networks and for planning of horizontal scaling.

5.1 Example 1 – Seamlessly Consolidating Multiple Subnets

This example illustrates the high availability and elasticity features of vWires. It shows how two subnets can be consolidated together without any change to the IP configuration of the machines. It also shows how this consolidation not only reduces the cost but also increases the availability of all existing services. Figure 5 represents the two independent subnets. To emphasize the elasticity point, the subnets use the same internal IP addresses.

The consolidation must meet the following two requirements:

- Existing IP addresses must be retained. Many services in the network such as firewalls, proxies, directory services, kerberos, and so on depend on IP addresses. Reassigning IP addresses during consolidation risks breaking down these services and therefore must be avoided.

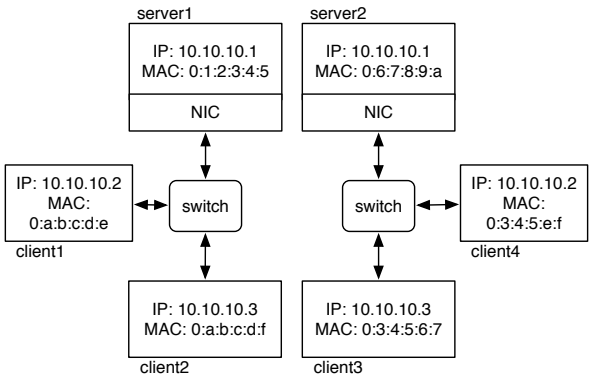


Figure 5: Example 1 – two separate physical subnets

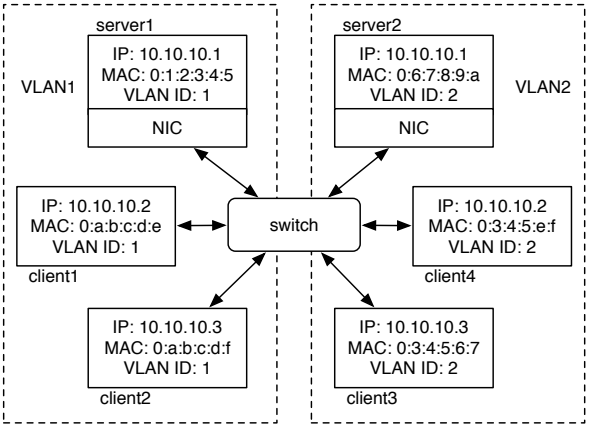


Figure 6: Example 1 – two VLANs sharing a physical network

- The consolidation must preserve the separation of traffic from the different subnets on the wire.

The traditional way to consolidate the two subnets on the same physical network would be to assign each subnet a VLAN ID, and then configure the switch ports with the appropriate VLAN IDs of the subnet. Finally, each machine is connected to the correct port. A VLAN-based network consolidation is represented in Figure 6. Note, however, that the resulting consolidation still retains the same number of machines and connections to a switch port.

A second approach would be to use virtualization. The two servers can be converted into two virtual machines that are co-hosted on a physical server. The same number of physical NICs for the two VMs can be retained, as well as the wire-port connectivity to the switch. From a hardware perspective, the redundancy of network connectivity ensures that there is no single point of failure.

The administrator has several options when assigning

NICs to the VMs. An obvious choice would be to assign the physical NICs, one to each VM. However, this option loses the advantage of high availability. In fact, the NIC of a specific VM becomes the single point of failure for that VM's network. If that NIC fails, then all the VMs behind that failed NIC become unreachable. Furthermore, this setup restricts the scalability of the configuration to the limited number of physical NICs that can be installed on the bus as well as the number of ports on a switch.

A better approach would be to first create a link aggregation that bundles the physical NICs together. The aggregation is then virtualized into multiple VNICs and assigned to their respective VMs. Figure 7 shows this virtualized consolidation. In Figure 7, the VNICs are created based on the VLAN ID of their respective VMs. Thus, even after the transformation to a virtual environment is completed, traffic from the different VMs can still be differentiated on the wire.

Furthermore, every VM benefits from the HA of the networking connectivity because it has a redundant path to the network. An outage of one of the NICs or its port on the switch will result in a possibly slower overall network, however each VM is still reachable.

We show below the steps needed to create the link aggregation and then the VNICs to create the configuration of Figure 7.

```
# dladm create-aggr -l nxge0 -l nxge1 \
aggr0
# dladm create-vnic -l aggr0 -v 1 vnic1
```

Note that in this example, the single switch constitutes a single point of failure. Switch stacking or layer-3 multi-pathing can be combined with link aggregations to provide high availability across multiple switches, as described in Section 3.6.

5.2 Example 2 – Consolidating Multi-Tier Enterprise Networks

This example is a typical scenario for a cloud operator that offers hosting services for its enterprise clients. Each client tenant of the cloud operator's data center expects complete separation from the other tenants. This example demonstrates that all the three tiers (web server, App server, Database server and iSCSI storage) of the client data center as shown in Figure 8 can move to the cloud but remain isolated and separate from other virtualized data centers in the cloud.

The following steps show how to convert one of the client enterprise's Intranets. First create the etherstub for the Intranet and three VNICs on top of it.

```
# dladm create-etherstub stub1
# dladlm create-vnic -l stub1 VNIC_WS1
```

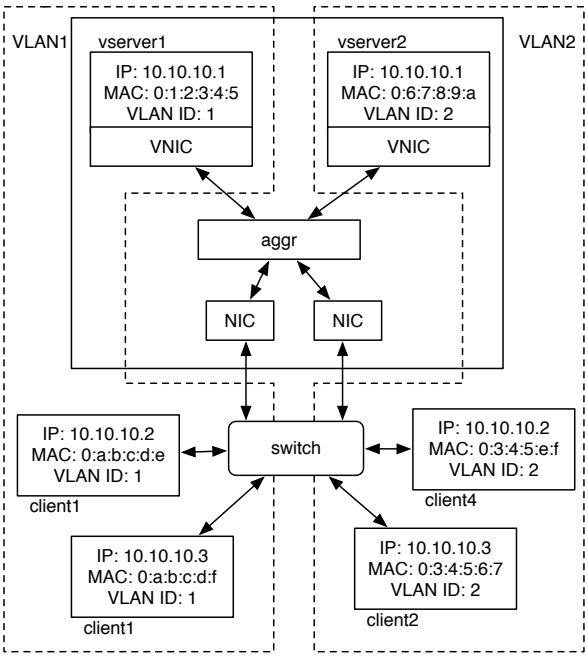


Figure 7: Example 1 – a vWire with two VLANs in a box

```
# dladlm create-vnic -l stub1 VNIC_AS1
# dladlm create-vnic -l stub1 VNIC_DB1
```

The VNICs can then be assigned to the zone `Webserver1` as described in Section 4.2.1. Similarly, assign `VNIC_AS1` and `VNIC_DB1` to `AppServer1` and `DBServer1`, respectively. Now connect the Database server to the back-end storage served by the iSCSI target: Create a VNIC on the back-end physical NIC:

```
# dladm create-vnic -l NIC2 VNIC_ST1
```

Assign `VNIC_ST1` to `DBserver1` as described in Section 4.2.1. Finally, connect the virtual enterprise subnet to the front-end edge router VNM by creating the `VNIC1` on the `Etherstub1` and assigning it to the Virtual Router VNM.

Figure 9 shows the resulting virtualized and consolidated Intranets for the two client enterprises. The physical servers have been converted into virtual appliances that are running in their respective zones. At the same time, the virtual network topology mimics the physical Intranets.

The two enterprises are competing for the CPU resources available on the virtualized server. Therefore, a remaining step is to define processor sets for each client, assign them to the zones, and bind the VNICs accordingly. Assume, for example, that `AppServer1` is assigned a processor set containing CPUs 1, 2, and 3. The VNIC can be bound to the CPUs assigned to `AppServer1` by issuing the following command:

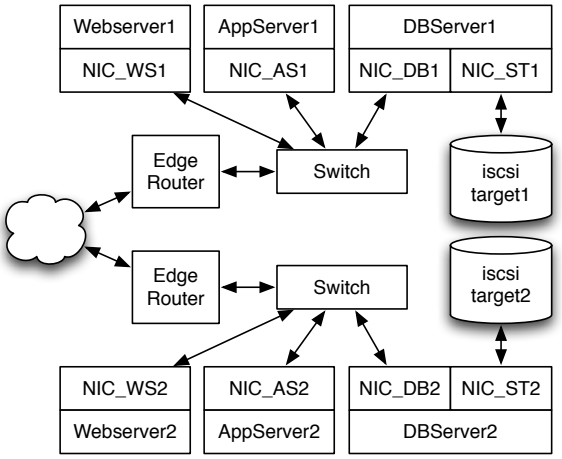


Figure 8: Example 2 – consolidating multi-tier enterprise networks, physical View

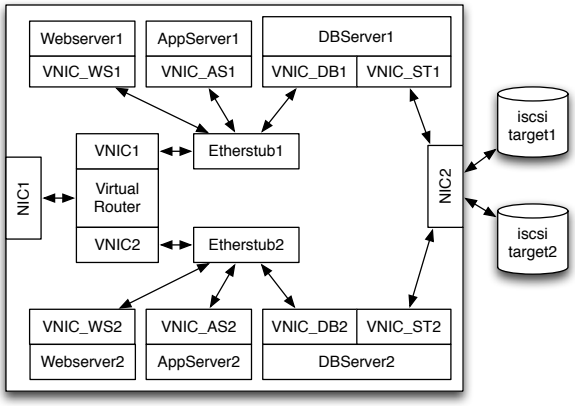


Figure 9: Example 2 – consolidating multi-tier enterprise networks, virtual View

```
# dladm set-linkprop cpus=1,2,3 VNIC_AS1
```

Future improvements will allow the data links to be automatically bound to the CPUs that are assigned to the zone, without requiring the administrator to manually bind the CPUs as shown above.

5.3 Example 3 – Try-Before-Deployment and Scale Out Scenario

In this example, we show how some of the observability and virtualization features of Crossbow can be employed to plan for scaling up the physical configurations as the need grows. The starting point is a small web server represented in Figure 10. As long as the amount of transactions coming from clients over the Internet is low, a single server is capable of handling the level of load required.

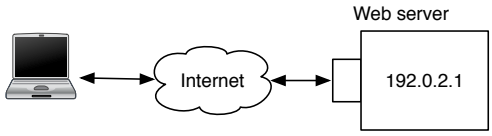


Figure 10: Example 3 – initial setting

In this scenario, the monitoring tools described in Section 4.3 can be used to log the usage history on the NIC to which the IP address 192.0.2.1 is associated:

```
# acctadm -e basic -f /var/log/net.log net
```

At this stage, only basic accounting for the networking interface is captured, and no flows are required. As the business picks up, the web server receives an increasing number of hits. A simple report to indicate the increased traffic activity can be obtained thus:

```
# dlstat -u -f /var/log/net.log
LINK    IBYTES  OBYTES  BANDWIDTH
e1000g    2.5M    0.1G 200.4 Mb/s
```

Anticipating further increase of traffic, the administrator can plan to horizontally scale the network up to multiple servers. However, before actually investing or committing any new physical resources to the network, it is desirable for the administrator to first understand how the new network configuration would actually behave while handling increased traffic. With Crossbow, the new distributed environment can be deployed and tuned in a virtual environment first.

In the give scenario, the web server is first virtualized into multiple virtual server instances running inside zones. Each instance can handle any of the URIs originally served. The virtual servers are connected to an in-box virtual switch through their respective VNICs. A load balancer and NAT appliance translates the IP addresses before forwarding the packet to the appropriate virtual server. An integrated load balancer [1] is expected to be available in OpenSolaris late 2009. Figure 11 shows the virtualized topology.

With the network usage history logging is still enabled, the amount of traffic on each link on the virtualized server can be monitored[2]:

```
# dlstat -u -f /var/log/net.log
LINK    IBYTES OBYTES  BANDWIDTH
```

[2]It is understood that most web servers also include logging of access statistics per URL. The authors' point here is to show how network infrastructure tools can be used for such accounting, whether the service being deployed included internal logging or not.

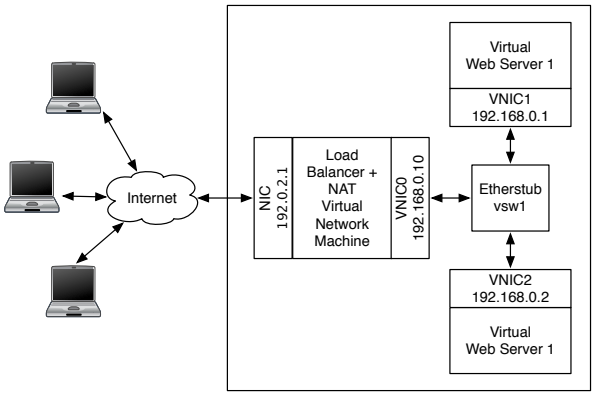


Figure 11: Example 3 – vWire for live workload analysis

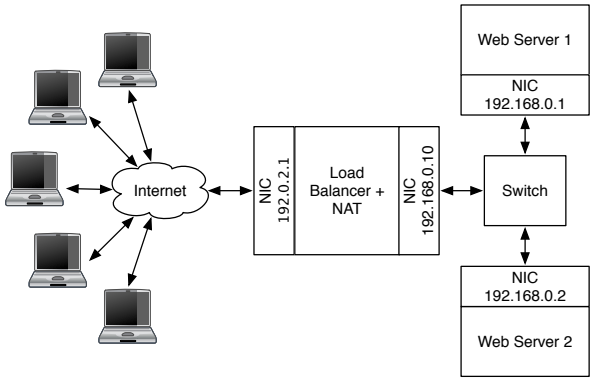


Figure 12: Example 3 – De-virtualizing for horizontal scaling

```
e1000g0    2.5M    0.1G   180.4 Mbps
vsw1       1.5M   52.7M   203.4 Mbps
vnic1      0.1M    3.0M    47.4 Mbps
vnic2      1.4M   49.8M   156.0 Mbps
```

This test run shows that the balance of traffic between the two virtual server appliances is imbalanced. The traffic through `vnic1` is only 23% of all traffic coming in the system, as opposed to the 77% being handled by the second virtual web server. The system administrator can then adjust the load balancer parameters to bring a more equitable distribution of the load.

When the load nears saturation levels for a single physical server to handle, the administrator can make an educated decision on the configuration of the new hardware. Note that the virtual web servers can be migrated to the new physical host with the exact same network configuration, without any need for IP renumbering. The final deployment is represented Figure 12.

It should be noted that more information can be derived from the usage history. The administrator could for example quantify the variation of load over time, and study the peaks of load, and the progression of the network usage, and extrapolate that progression to estimate the right time to start considering an upgrade.

6 Related Work

The Crossbow architecture provides mechanisms to achieve network virtualization within a host with ease of use and minimum performance penalty. The virtual NICs and flows leverage NIC hardware advancements such as classification and multiple receive and transmit rings to ensure the separation of virtualized packet streams without any processing overhead on the host. The virtual NICs and flows can be created over physical NICs, link aggregations, and etherstubs to provide private connectivity between virtual machines.

The idea of virtual switching has been implemented in other main stream virtualization technologies as well. Citrix System Xen [7] has a native Linux implementation where the physical NIC is owned by the hypervisor and virtual machines access the network by means of a front end driver that run in the guest domain and the back end driver that runs in the hypervisor. The hypervisor runs the physical NIC in promiscuous mode and uses a software based bridge implementation to provide all packets to the back-end drivers, which then select the packets that match their respective MAC addresses. There are mechanisms available to enforce bandwidth limiting and firewall rules on the traffic for virtual machines. However, these are typically separate subsystems, often very complex in implementation and administration, and can result in significant performance overheads [25]. VMware ESX based hypervisor has a proprietary implementation on a Linux variant but apparently suffers from some of the same issues [26] in terms of demultiplexing packets for various virtual machines and resource separation.

More recently, Cisco Systems announced a new virtualization offering under the Unified Computing System (UCS) [22] umbrella and based on the VMware EX hypervisor. The solution uses a specialized NIC along with a Nexus switch where packets from individual virtual machines are tagged to allow the switch to implement virtual ports and provide features similar to the Crossbow implementation. A centralized management solution in the form of a Virtual Supervisor module manages the physical and virtual components on the switch as well as hosts to provide easy management of resources and filtering policies. At the same time, the implementation is proprietary to Cisco software and hardware and VMware ESX hypervisor.

Some work is also occurring in the research community as part of the OpenFlow [11] consortium which helps in building a standard based programmable switch.

Such a switch would enable the Crossbow based hypervisor to program the switch with VLAN tags that are associated with customers and thus create more dynamic virtual networks where the switch can also provide separation, fairness, and security for the Crossbow vWire.

7 Conclusion and Future Work

The Crossbow virtualization and QoS components presented in this paper provide a unique mechanism to achieve network virtualization and consolidate multiple networks into one physical network. Assigning VLAN tags to VNICs and performing host based VLAN switching allow the creation of fully virtualized and isolated networks. Because the VNICs can be assigned link speeds, priorities, and dedicated NICs and CPU resources, a collection of virtual machines can span multiple physical machines and yet have deterministic performance characteristics. The configuration of VNICs and resource assignment is easy to configure and can be driven by external management tools with the provided APIs.

Apart from VNICs and virtual switches, multiple VNICs on different physical NICs can be assigned to OpenSolaris zones or virtual machines to create network components like routers, load balancers, firewalls, and so on. These virtual network machine along with VNICs and virtual switches can be combined together to create a fully virtualized network called vWire.

The Crossbow vWire offers a fully elastic, isolated, and dynamic virtualized network where virtual machines can migrate to other physical machines. The vWire extends with these VMs without needing any changes to the physical cabling or switches. Since the vWire uses VLAN tags and extended VLAN tags to provide isolation, it can work with any existing switch.

The various enterprise level features for failover and high availability such as link aggregation and IPMP, are designed in the architecture. Thus VNICs can be created over link aggregations and multiple VNICs on different attach points can be assigned to the same IPMP group. Care has been taken to ensure that a VNIC shows up as a separate interface on the MIB with the configured link speed as the interface speed. Existing network management tools can thus continue to work seamlessly in a virtualized environment.

The various examples in this paper show some of the possibilities where Crossbow can be used in an enterprise to decouple the application from the physical hardware and network to ensure easier deployment, management, and hardware upgrade. Because the vWire is a collection of rules and objects, it can be easily migrated from one physical network to another. This flexibility allows enterprises to migrate their network in full or in part to a public cloud when needed. The same concepts can be used by startups to create their data-center in a box in a public cloud. They can use Crossbow tools to analyze their usage and scale out to multiple machines seamlessly as business needs and traffic grow.

The core of the Crossbow architecture and all the features described in this paper have been implemented and integrated in OpenSolaris and available at http://opensolaris.org to any user.

Near term work focuses on enhancing the management tools to visualize and configure these vWires and virtual network machines. Crossbow has achieved a powerful level of control and observability over the networking resources inside a single system. One of the directions being pursued is to extend that kind of control beyond the boundaries of a single box, to encompass flows that span multiple subnets of physical and virtual machines. To that end, new wire protocols are being explored to convey some of the QoS requirements between nodes. We need to address both the data plane, and the control plane. Priority-based Flow Control (PFC) is the layer-2 mechanism defined by the IEEE and used for discriminating based on the VLAN tag's priority field on data packets. On the control plane, Generic Attribute Registration Protocol (GARP) and Multiple VLAN Registration Protocol (MVRP) are being considered for two reasons: The scalable administration of multiple interconnected nodes underscores the need for a hands off propagation of QoS information across the links. Secondly the network must be protected from the floods of unnecessary broadcasts from unused VLANs.

8 Author Biographies

Sunay Tripathi is a Distinguished Engineer at Sun Microsystem working on networking and network virtualization. He received a MS in Computer Science from Stanford University in 1997. His blog is at http://blogs.sun.com/sunay, and he can be reached at sunay.tripathi@sun.Com

Nicolas Droux is a Senior Staff Engineer and architect with the Solaris Core OS group at Sun Microsystems. Nicolas has led, designed, and implemented several kernel projects in the areas of High Performance Computing, I/O, security, virtualization, and networking. His blog is at http://blogs.sun.com/droux, and he can be reached at nicolas.droux@sun.com.

Kais Belgaied is a senior staff engineer and a technical leader at Sun Microsystems, Inc. His areas of interest include networking, virtualization, operating systems, cloud computing, and IT security. He is a voting member of the Platform Architecture Review Counsel with Sun Microsystems, and an active participant in multiple IETF

working groups. His blog is http://blogs.sun.com/kais, and he can be reached at kais.belgaied@sun.com.

Shrikrishna Khare is a Solaris Kernel Networking engineer at Sun Microsystems. He received a M.S. in Computer Science from North Carolina State University, USA in 2008. He can be reached at shrikrishna.khare@sun.com

References

[1] http://opensolaris.org/os/project/vnm/ilb (PSARC/2008/575).

[2] http://www.virtualbox.org.

[3] http://www.vmware.com/pdf/virtualization.pdf.

[4] M. Allman and W. S. V. Paxson. TCP Congestion Control. In *RFC 2581*, 1999.

[5] W. Almesberger. Linux Network Traffic Control.

[6] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the Clouds: A Berkeley View of Cloud Computing. In *Technical Report No. UCB/EECS-2009-28*.

[7] P. Barham, B. Dragovic, K. Fraser, T. H. Steven Hand, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. In *19th ACM symposium on Operating System Principles*, pages 164–177. ACM, 2003.

[8] Carson. Limiting Bandwidth Usage on Xen Linux Setup.

[9] J. Crowcroft, S. Hand, R. Mortier, T. Roscoe, and A. Warfield. QoS's Downfall: At The Bottom, or Not at All! In *RIPQOS'03: Proceedings of the ACM SIGCOMM workshop on Revisiting IP QoS*, 2003.

[10] Intel. Intel 82598 10GbE Ethernet Controller Open Source Datasheet, 2008.

[11] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Enabling Innovation in Campus Networking. 2008.

[12] Neterion. Neterion Xframe II 10 Gigabit Ethernet.

[13] L. M. S. C. of the IEEE Computer Society. IEEE Standards for Local and Metropolitan Area Networks: Virtual Bridged Local Area Networks. In *IEEE Std 802.1Q-1998*, 1998.

[14] D. Price and A. Tucker. Solaris Zones: Operating System Support for Consolidating Commercial Workloads. In *18th Large Installation System Administration Conference*, pages 241–254. USENIX, 2004.

[15] T. H. Ptacek and T. N. Newsham. Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection. White paper, 1998.

[16] C. L. Schuba, I. V. Krsul, M. G. Kuhn, E. H. Spafford, A. Sundaram, and D. Zamboni. Analysis of a Denial of Service Attack on TCP. In *In Proceedings of the 1997 IEEE Symposium on Security and Privacy*, pages 208–223. IEEE Computer Society Press, 1997.

[17] M. Seaman. A Multiple VLAN Registration Protocol (MVRP), 2003.

[18] S. Soltesz, H. Potzl, M. Fiuczynski, A. Bavier, and L. Peterson. Container-Based Operating System Virtualization: a Scalable High-Performance Alternative to Hypervisors. In *2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, pages 275–287. ACM, 2007.

[19] Sun Microsystems. Solaris 10 System Administration Guide: IP Services, 2008.

[20] Sun Microsystems, Inc. Sun Multithreaded 10GbE (Gigabit Ethernet) Networking Cards, 2007.

[21] Sun Microsystems, Inc. System Administration Guide: Solaris Containers-Resource Management and Solaris Zones, 2009.

[22] C. Systems. Unified Computing Systems.

[23] S. Tripathi, K. Belgaied, and N. Droux. Crossbow: Network Virtualization Resource Partitioning.

[24] S. Tripathi, N. Droux, T. Srinivasan, and K. Belgaied. Crossbow: From Hardware Virtualized NICs to Virtualized Networks. In *In Proceedings of the ACM SIGCOMM Workshop VISA'09*, 2009.

[25] S. Tripathi, N. Droux, T. Srinivasan, K. Belgaied, and V. Iyer. Crossbow: A Vertically Integrated QoS Stack. In *In Proceedings of the ACM SIGCOMM Workshop WREN'09*, 2009.

[26] J. S. G. Venkitachalam and B. Lim. Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor. In *Proceedings of the 2001 USENIX Annual Technical Conference*, 2001.

EVA: A Framework for Network Analysis and Risk Assessment

Melissa Danforth
Department of Computer Science
California State University, Bakersfield
Bakersfield, CA 93311
mdanforth@csub.edu

Tags: security, research, attack graphs

Abstract

EVA is an attack graph tool that allows an administrator to assess and analyze a network in a variety of fashions. Unlike other attack graph tools which just focus on visualizing the network or recommending a set of patches to secure the network, EVA goes beyond these modes to fully explore the power of attack graphs for a multitude of administrative and security tasks. EVA can be used to derive a set of hardening measures for a network, to perform strategic analysis of a network, to design a more secure network architecture, to assist in forensic evaluations after a security event and to augment an intrusion detect system with information about the likely targets of an attack. This paper summarizes the framework used by EVA, provides real-world results of using EVA and shows how EVA is scalable to large networks.

1 Introduction

Securing a computer network against intrusion is a complicated task. The risk profile of the network depends not only on the configuration of individual machines, but also on the connectivity between machines. If an administrator only evaluates the risk profile of each machine individually, he will miss multi-stage attacks that propagate across the network. For example, an attacker might compromise a public web server and then use that server to compromise the database server. This is a classic "foothold" scenario whereby the attacker compromises one machine to use as a base for gaining access to internal networks he could not directly access. Such scenarios must be considered when evaluating a network.

Attack graphs [2, 4, 10, 11, 14, 15, 16, 17, 18, 19, 9, 21] and attack trees [6, 7] provide a method to discover and visualize such "foothold" scenarios in the network. Attack graphs and trees compute exploit paths that a theoretical attacker might take through the network, given knowledge of the vulnerabilities on each machine, the firewall rules in the network and the topology of the network. Attack graphs by themselves are purely just a method to represent and possibly visualize these paths. The true power of attack graphs lays in analyzing the attack graph.

EVA (Evolutionary Vulnerability Analysis) is an attack graph tool that supports a multitude of analysis modes. As shown in [5], it is scalable to large networks containing hundreds of hosts. This paper describes further improvements that increase the scalability to networks containing thousands of hosts. EVA is a policy driven model, which allows administrators to tune the analysis to the specific operating criteria or mission for their networks. The policy model is flexible so that the administrator does not need to provide extensive information to it.

Most prior work has focused on two modes of analysis: finding a set of hardening measures and performing "what if" scenarios. A set of hardening measures are typically patches or firewall rules that prevent the attacker from achieving one or more goals. The "what if" scenarios allow the administrator to pretend there are unknown vulnerabilities in the network. This allows an administrator to explore the consequences of unknown vulnerabilities, such as "zero-day" exploits. The "what if" mode essentially alters the input into the attack graph tool to support the scenario instead of the actual network. The resulting "what if" attack graph that can be analyzed using other modes of analysis. EVA supports these modes of analysis and uses the policy to guide the analysis.

EVA goes beyond these modes of analysis to further unlock the power of the attack graph model. It can also be used for network design, forensic evaluation and IDS monitoring. For the network design mode, the tool can be used in two ways. First, it can be given multiple prototype networks to evaluate and decide which has the best

security. The mode of analysis has also been used in GARNET [21]. The second use of the tool for network design is unique to EVA. Given a prototype network, it can automatically alter the connectivity and/or add IDS sensors to improve the security of the network. As with hardening measures, this analysis is guided by the policy for the network.

For forensic analysis, the evidence gathered during the course of the investigation is given to the tool. The tool then produces a list of resources that the attacker could have also compromised given the evidence. This gives direction to the forensic evaluators by pointing out likely paths the attacker took during the compromise. IDS monitoring uses a similar approach, but in real-time as opposed to after-the-fact. Theoretically, the list of potential exploit paths could be given to a intrusion response system to prevent the attacker from actually exploiting those paths.

These analysis modes have not been explored in other attack graph tools. This work describes how EVA can be expanded to supporting these new analysis modes. By supporting these modes, EVA has a much wider use than simply visualizing or securing the network. It can be used in multiple phases of operation for a variety of security purposes.

Section 2 describes prior works in attack graphs and attack trees. This section highlights how EVA differs from these prior works. Section 3 details the attack graph model used by EVA. In Section 4, the methodology used to generate the attack graphs is given. Section 5 describes the genetic algorithm used for analyzing attack graphs. Section 5 also details the policy model and the various modes of analysis. Section 6 provides some experimental results of using EVA on our student lab network and on simulated networks. Section 7 talks about future work to improve this tool.

2 Related Work

Several prior works [11, 15, 17, 18] have shown that determining a set of hardening measures is in NP. Philips and Swiler [15] also shows that the problem of placing sensors to maximize coverage of the exploit paths an attacker could take is in NP as well. Given this, most prior works have focused on non-adaptive approximation methods to find a set of hardening measures.

Philips and Swiler [15] allow an administrator to secure one resource at a time by computing shortest paths to that resource. This does not actually provide a set of hardening measures, but instead trims the attack graph to just the most likely paths an attacker would take. Their method requires extensive administrator interaction to actually determine the hardening measures and to secure all the resources on the network.

Other groups have proposed non-adaptive approximation methods to derive a set of hardening measures. Noel, *et al.* [10, 14] derive an algebraic expression of the initial conditions that allow an attacker to compromise a single resource. Sheyner, *et al.* [11, 17, 18] use a greedy algorithm to protect a given resource. Ammann, *et al.* [2] compute the hardening measures for a single resource based on information added to each node during the attack graph generation. These methods only compute the set of hardening measures for a single "goal" at a time. They must be repeated for each resource the administrator wishes to protect. This requires not only more processing time, but most likely will result in repeating computational steps when two resources share a portion of their exploit paths. EVA on the other hand derives a set of hardening measures to protect all the resources the administrator has marked as critical.

Dewri, *et al.* [7] uses a genetic algorithm to compute a set of hardening measures for one or more resources. Their algorithm also supports each hardening measure having a different cost. This is similar to the approach used by EVA, but there are several critical differences, as detailed in [5]. First, their cost model is not very flexible. It requires the administrator to assign a cost and weight for every single possible hardening measure. Since the number of hardening measures increases dramatically as the size of the network increases, Dewri's method would require extensive user input before being able to compute the set of hardening measures for larger networks. EVA uses a default cost for most hardening measures, but allows the administrator to adjust the cost for any hardening measure. The administrator also has flexibility in this adjustment. One can adjust a measure globally, such as "do not allow port 80 to be disabled", or one can adjust a measure for a specific machine. Thus, the administrator only has to specify costs for those measures deemed desirable or undesirable for the network.

Second, the genetic algorithm used in [7] is not very scalable to large networks, as shown in [5]. This is because they use a multi-objective genetic algorithm that treats the security provided by the set of hardening measures and the cost of that set as equals. As shown in [5], this leads to their algorithm maintaining a set of low cost but also low security hardening measures. Most of these low cost solutions turn out to be evolutionary dead-ends because they provided very little security. By maintaining them, the genetic algorithm in [7] is essentially wasting memory and computational time on untenable solutions. The genetic algorithm used by EVA uses a priority based method which first prioritizes on securing the network and then looks at minimizing the cost of the set of hardening measures. The experimental results shown in [5] show that this is a far more suitable approach for the attack graph problem.

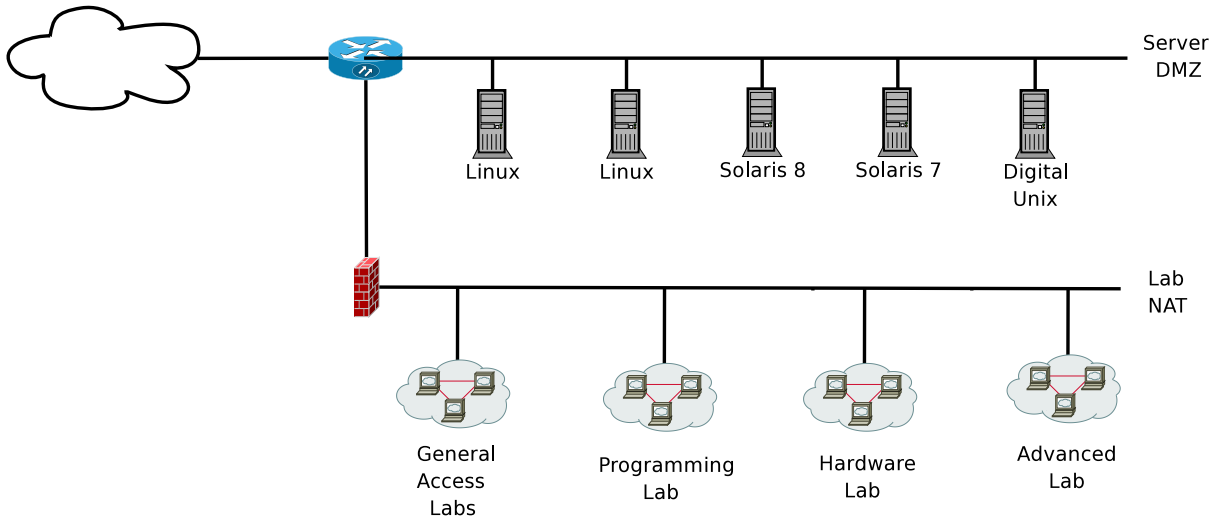


Figure 1: The Computer Science instructional network that was scanned for modeling in EVA.

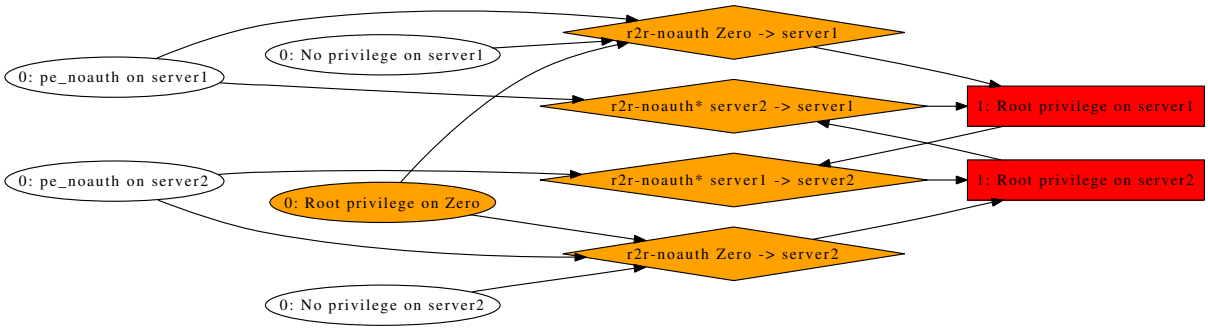
NetSPA [9] and its graphical front-end GARNET [21] are the closest competitors to EVA in the market today. NetSPA is a project out of MIT Lincoln Labs that was awarded $10k in MIT's 2008 Entrepreneurship Competition to form a startup company based around NetSPA called CyberAnalytix [13]. While NetSPA is similar to EVA, there are several key differences between NetSPA and EVA. First, NetSPA uses a different technical approach to the attack graph problem than EVA. NetSPA focuses on the data structure of the attack graph and post-processing the attack graph to reduce complexity. EVA uses a classic adjacency-list data structure for the attack graph and focuses on pre-processing the network using an abstract exploit model described in Section 3.1 and a meta-machine model described in Section 3.2 to reduce the complexity of the network. Both approaches provide scalability, but are fundamentally different in nature. Second, NetSPA uses a non-adaptive algorithm to compute the set of hardening measures while EVA uses an adaptive genetic algorithm that incorporates the site's policy when computing the set of hardening measures. By incorporating the policy, EVA is able to provide recommendations tuned to the site's mission or operating criteria. Third, NetSPA and GARNET focus on providing a set of hardening measures and visualizing the network for both actual networks and theoretical ("what if") scenarios. EVA supports these modes and also adds modes for network design, forensic evaluation and IDS monitoring. This gives EVA more versatility.

3 Attack Graph Model

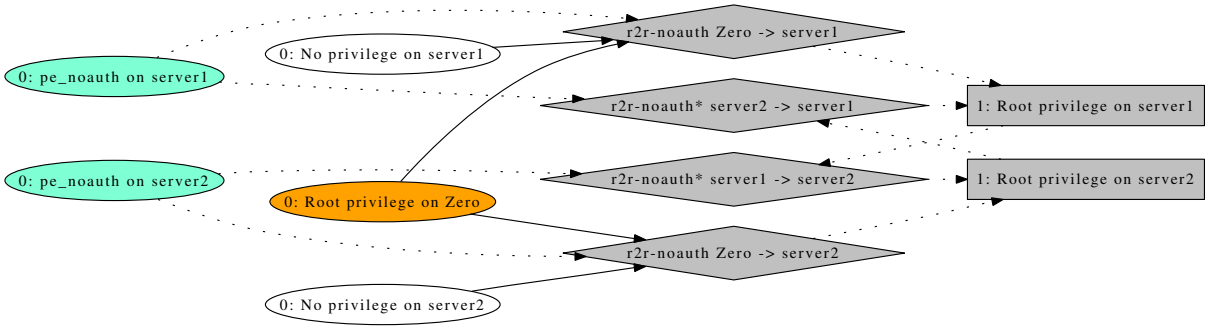
The attack graph model used by EVA was first described in [4]. The attack graph itself is an adjacency-list matrix that describes the exploit paths an attacker could take through the network. The inner nodes of the graph represent various states the attacker has achieved, such as "user privilege on $host_5$". The initial nodes of the graph represent the initial state of the network, such as what vulnerabilities are present and what privileges the attacker has initially. The edges of the graph represent exploits the attacker has executed. An attack graph for the network evaluated in Section 6 is shown in Figure 2.

The primary underpinning of the model is a set of exploit templates that describe exploits an attacker could use in the network. These templates are represented in a "requires/provides" [12] format. The "requires" portion of the template specifies what conditions must exist for the exploit to occur. The "provides" portion of the template states the consequences of the exploit, such as new privileges the attacker gains from the exploit. An attack graph is built by matching templates to the current knowledge about the network. When all of the "require" conditions are met for a template, it is executed and all of its "provide" conditions are added to the attack graph. For purposes of the representation, each condition is tied to an individual node in the attack graph.

The initial nodes of the attack graph are derived from several sources: a model of the network connectivity, a list of vulnerabilities present on all machines in the network and an attacker model. The model of the network connectivity describes the firewall and/or routing rules that would prevent two hosts from communicating with one another. By default, EVA assumes two hosts can



(a) Unpatched



(b) Patched

Figure 2: The attack graph for the basic network configuration and the patched attack graph after analyzing it. The color scheme for the graphs is as follows. The orange oval is the attacker's starting point. Red boxes represent machines where the attacker has obtained root privileges. Yellow boxes are machines where the attacker has gained user privileges. Orange diamonds are the attacks executed against the network. Clear ovals are the initial conditions in the network. In the patched graph, disabled attacks and nodes are grey while the patched vulnerabilities are aqua. For this network, the recommended patches prevent the attacker from getting root on both machines, since both root nodes have been disabled in the patched graph.

communicate on a given port. The connectivity model only needs to specify denied connectivity. When referring to TCP connections, the denied connectivity is assumed to be a denied SYN packet. The denied connectivity is directional, just as firewall rules are. For example, if you deny $host_1$ from connecting to port 443 on $host_4$, this does not prevent $host_4$ from connecting to port 443 on $host_1$. This connectivity matrix can be derived from knowledge of the network's firewall and routing rules.

The list of vulnerabilities on all machines in the network can be obtained via vulnerability scanner reports. The vulnerability name must match the naming structure used in the exploit templates. Currently, EVA can translate certain Nessus [20] plugin IDs to a vulnerability name. The machines must also be given unique names in this list and these names must match the names used in the connectivity model. IP addresses or domain names are a very logical name to use for this model. One can also rename machines in both the vulnerability list and connectivity model to any name of the administrator's choosing.

The attacker model describes what initial privileges the attacker has in the network and where the attacker is located in the network. For example, one can model an attacker that is outside of the network and who has no initial privileges in the network (the "outsider" problem). One can also model an attacker who has a machine inside the network under his control or who has certain privileges inside the network (the "insider" problem). One can also use any combination of these two problems, although currently EVA assumes a single-attacker model, so it cannot distinguish between the nodes achieved by two or more attackers. In other words, if you model both an insider and outsider, EVA assumes them to be the same person. Allowing multiple attackers is a planned future refinement (see Section 7).

One of the issues with attack graphs that was described in more detail in [4] is that the number of edges in the graph, in other words the number of exploits executed by the attacker, is dominated by the number of exploit templates in the model and the number of machines in the network. If a is the number of exploit templates and n is the number of machines, then the number of edges is $O(an^2)$. To achieve scalability, one must reduce the number of exploit templates and the number of machines in the network in such a way that it does not affect the functionality of the attack graph. To do this, EVA uses two approaches: an abstract model of exploit classes and clustering of identical machines.

3.1 Abstract Model of Exploit Classes

When looking at the early literature on attack graphs, particularly the work of Sheyner, *et al.* [11, 17, 18], two things became clear. First, if one were to model each and every exploit that existed in the world, the exploit templates would quickly grow to an enormous size. Second, many exploits shared characteristics and only varied by the name of the vulnerability and/or the port number used in the exploit. One could greatly reduce the number of exploit templates required by coming up with abstracted templates that apply to a variety of actual exploits.

The difficulty with this approach is creating abstract templates that retain the ability to model different types of exploits while still grouping multiple exploits together. Essentially, a classification system had to be developed for exploits. The details of this classification system are given in [4]. In brief, exploit classes such as "remote to user" or "remote to root" were developed. Most of the classes focus on privilege escalations, client-side privilege escalations (such as a browser exploit), username/password guessing, password cracking, information leaks, bypassing firewall rules or altering router rules. The model currently does not support denial of service, but it could be extended to do so by writing a new set of rules for that class.

Rewriting the exploit templates is only part of the abstraction process. The vulnerability list also must be translated from actual vulnerabilities to abstract vulnerabilities. This is done currently by comparing the Nessus [20] plugin ID to a mapping that converts known Nessus plugin IDs to their corresponding abstract vulnerability. This mapping is currently maintained by hand. The translation of the vulnerability list is done during the pre-processing stage, before generating the graph. For each machine in the vulnerability list, its set of vulnerabilities are translated to the abstract vulnerability class. If two or more vulnerabilities for that machine map to the same abstract class, the duplicates are discarded. When post-processing the reports generated by the analysis tool, this process is reversed.

Likewise, the port numbers given in the model of network connectivity must also be abstracted. This is a slightly more complex process, since any given port may be used for more than one abstract exploit class. Again, a mapping of port number to abstract port name is used, except this mapping supports one-to-many mappings where one port number might be associated with several abstract exploit classes.

There are two major advantages to having an abstract model for the exploit templates. The first advantage is that this greatly reduces the size of the template set. By reducing the number of templates, the number of edges in the graph are also reduced, as detailed above. This increases the scalability of the model since, as described in [4], the number of edges are a prime indicator of the complexity of the attack graph. The second advantage is reduced administrative overhead. One does not have

to alter the exploit templates every time a new exploit comes to light. Instead, the administrator can see if that exploit is part of an existing abstract class. If so, the preprocessing mappings can be altered to support this new exploit. If not, the model allows an administrator to write templates for specific exploits that are not covered by the abstract templates.

3.2 Clustering

The second approach to reduce complexity and increase scalability is to group identical machines into a cluster. In [4], this cluster was modeled as one meta-machine. This has been updated to model each cluster as two machines, so that the interactions between machines in a cluster can be observed.

The process of clustering is similar to what was described in [4]. After the connectivity model and list of vulnerabilities has been pre-processed for the abstract template model, it is further pre-processed to discover the clusters. On the first pass, all machines with identical vulnerabilities are put into a proto-cluster. On the second pass, each proto-cluster is subdivided into the final clusters based on the connectivity. Each final cluster contains machines with identical vulnerabilities and identical connectivity. Each cluster is assigned a name and the members of that cluster are recorded. Then the vulnerability list and connectivity model are updated as follows. If a cluster contains only one machine, that machine is left as-is in both the vulnerability list and the connectivity model. If a cluster contains two or more machines, all machines in the cluster are removed from both the vulnerability list and connectivity model. Then two machines whose names are based on the cluster name are added to both the connectivity model and vulnerability list. These two cluster machines have all the vulnerabilities and connectivity rules specified by the original machines in the cluster. Clustering is currently done with a Perl script to parse and alter the input files.

For a network which has large segments of identical machines, clustering can greatly improve the performance of EVA by reducing the number of machines modeled in the attack graph. Since the members of the cluster are recorded, it is easy in post-processing to augment all reports about a cluster with the list of machines in that cluster. The administrator can then tell that hardening measures need to be applied to all machines in the cluster.

4 Generation of Graphs

As described in Section 3, the exploit templates are in a "requires/provides" format. This makes them well-suited to be encoded as rules in an expert system. The expert system JESS [8] is used by EVA. The abstract exploit templates are encoded as rules in the expert system. These rules use the CLIPS [1] syntax, so the ruleset could be exported to other expert systems that support this syntax. The network connectivity model, the list of vulnerabilities and the attacker model are encoded as initial facts to the expert system. From these initial facts, the "requires" portion of zero or more templates is satisfied. The "provides" portion of the template asserts more facts into the expert system. This in turn may satisfy other templates.

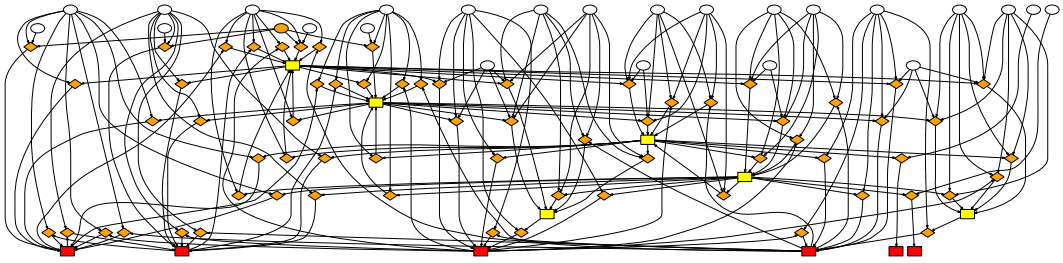
Unlike some prior works [11, 17, 18, 16] which only see if the attacker can achieve a specific goal, such as "get root on the web server", EVA uses an exploratory approach to seek out all possible exploit paths the attacker could take through the network. The matching of facts to exploit templates continues until the newly asserted facts cause no more templates to be satisfied. Thus all avenues of attacks that can be described given the initial facts and the exploit templates are explored.

The expert system also records each exploit template rule that is activated, the facts that caused it to be satisfied and the facts that are asserted as a consequence of it being activated. This is equivalent to one edge in the attack graph. The nodes in the attack graph are equivalent to the facts in the expert system, which are also recorded. A Perl script translates the output of the expert system into two formats: a visualization format and the genetic algorithm format. The visualization format uses the DOT syntax of the Graphviz project [3]. From DOT, one can produce images in a variety of formats such as EPS and GIF. The genetic algorithm format is a list of annotated edges used to construct the adjacency-list matrix for analysis.

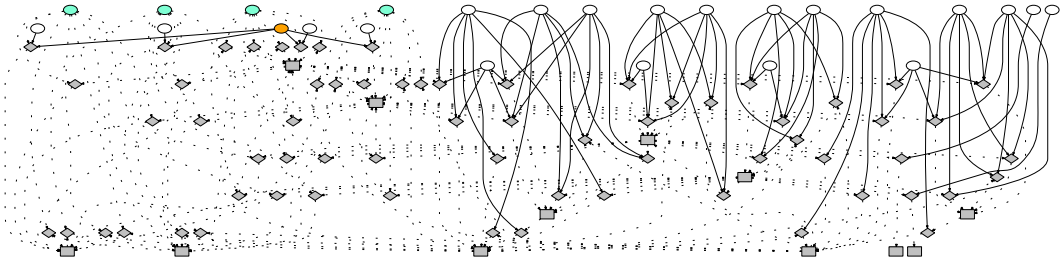
5 Evolutionary Analysis

In order to determine a set of hardening measures, one must first specify what is considered to be the "bad" states in the attack graph, i.e. what the administrator does not want the attacker to achieve. For example, the administrator might want to prevent the attacker from gaining root-level privileges on all hosts. When deriving the hardening set, one then seeks to disconnect the attacker from these undesirable states by applying a hardening measure. The "bad" states correspond to a set of nodes in the attack graph. This can be given specifically, such as "prevent root access on $host_8$", or generally, such as "prevent root access on all hosts". These bad states are referred to collectively as the goal nodes since they represent the goals of the attacker.

Related to finding a set of hardening measures, one can also analyze the network to assess its risk profile. To do so, one simply measures how many of these "bad"



(a) Unpatched



(b) Patched

Figure 3: The attack graph and analyzed attack graph for the scenario where a user visits a malicious website with a vulnerable web browser. This is a classic outsider scenario where the attacker gains a foothold in the network then uses this foothold to further compromise the network. The color scheme is as described in Figure 2.

states the attacker has obtained and output that as a risk metric or a risk profile. Again, this can be tuned to the particular needs of a given network by changing the set of "bad" states to reflect what is undesirable for that particular network.

The hardening measures supported by EVA are patching a vulnerability, adding a firewall rule and placing an IDS sensor. Priority is given to each hardening measure based on the policy model and the mode of analysis. Each measure has two attributes associated with it: the cost of that measure and the security provided by that measure. Both attributes can be manipulated by the policy and by the mode of analysis. When the mode of analysis is to derive a set of hardening measures, the default costs in order from cheapest to most expensive are patches, firewalls and IDS sensors. The default behavior is to have patches and firewall rules confer more security than IDS sensors. Any of these defaults can be changed by the policy model. One can also tell the genetic algorithm to only consider a subset of hardening measures, such as to just consider patches.

A genetic algorithm was chosen as the means of doing the analysis. As described in [5], finding a set of hardening measures directly is computationally infeasible. One cannot "brute force" the solution. Genetic algorithms are an approximation method that allows one to start with random solutions and then refine those solutions into better solutions via an evolutionary process. This is essentially a guided search of the solutions space. Each solution is referred to as a chromosome. A group of solutions being evaluated are called a population. The evaluation continues iteratively for several rounds, with each round being called a "generation". Initially, in the first generation, the population is randomly generated. Then the "fitness" of each chromosome is evaluated. The fitness function determines how well a given solution works for the problem. The most fit chromosomes are then selected as parents and recombined, with the hopes of creating even better solutions. Finally, a few chromosomes are randomly mutated. In EVA, a mutation flips the bit, so if a hardening measure was in use, it would no longer be used and vis versa. After recombination and mutation, the population moves on to the next generation, where it begins with evaluating the fitness of the chromosomes. The population will keep passing through the fitness evaluation, recombination and mutation steps until the programmed maximum number of generations has elapsed.

More details about the genetic algorithm can be found in [5]. The code has been updated since that time to be

multi-threaded when evaluating the fitness of the population. Since each chromosome in the population has its own fitness, this point of the evaluation is well-suited to multi-threading. The population is broken down into sub-groups and each sub-group spawns a thread to evaluate the fitness of the chromosomes in its sub-group. The number of threads is selected when the program is compiled. Currently, four threads are spawned. The main program waits for each thread to complete before moving on to the recombination step.

The chromosome in the genetic algorithm corresponds to a proposed set of hardening measures. During fitness evaluation, each measure in the chromosome is applied to the attack graph. Each node and edge in the attack graph records how it is affected by the measure. A patch disables an initial node, which corresponds to a vulnerability on a machine, and all edges leading out of that node, which correspond to attacks enabled by that vulnerability. A firewall rule disables an edge, which corresponds to the attack that the firewall rule blocks. An IDS sensor watches an edge. This indicates that the attack represented by that edge will be detected if it is executed. After applying the hardening measures, a cascade effect takes place throughout the graph, as described below.

Edges, which correspond to one specific attack, will disable themselves if any incoming node to that edge is disabled. This is because the incoming nodes correspond to preconditions required for the attack to succeed. If any precondition becomes disabled, the attack can no longer succeed, so the edge disables itself. It does not disable the other incoming nodes though since those have not been affected by the fact that the attack can no longer succeed. Similarly, if any of the incoming nodes for an edge are watched, the edge marks itself as watched. This indicates that one of the preconditions for the attack is enabled by an attack that the IDS can detect. This will only occur when several attacks are needed in order for the attacker to reach a goal. While the IDS may not detect the attack corresponding with this edge, it has detected an early attack that is required for this edge's attack to succeed. Thus, this edge will mark itself as watched.

Internal nodes will disable themselves when all their incoming edges are disabled. This means that all attacks which lead to that state have been disabled. When a node disables itself, all edges leading out from that node will disable themselves due to the behavior of edges described above. Similarly, when all edges coming into a node are watched or disabled, the node will mark itself as watched. This indicates that all possible paths to the privilege or condition represented by the node have been covered by IDS sensors. The attacker cannot reach this node without triggering an IDS alarm, so the node is marked as watched. This will then trigger all edges leaving that node to mark themselves as watched, for the reasons described above. If a node or edge is marked as both watched and disabled, the disabled state takes priority.

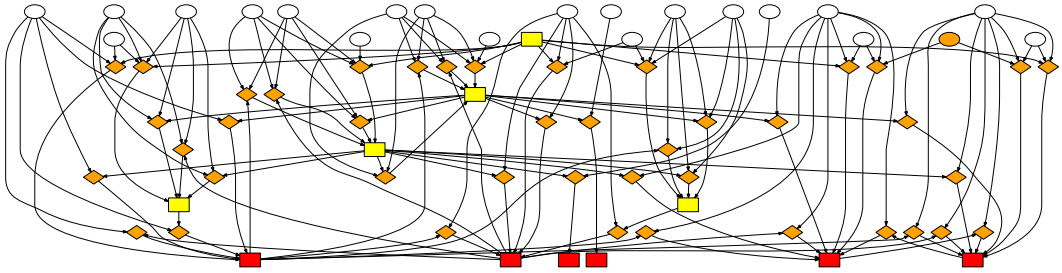
At the end of applying all the proposed hardening measures and this cascade effect, each goal node is checked. The preferable result is that all the goal nodes have been disabled. For each node that is not disabled, its risk metric is calculated based on if it is being watched by an IDS sensor and how many enabled edges can still reach it. The sum of the risk metrics for each goal node is the overall risk that is still present with that proposed set of hardening measures. The genetic algorithm fitness function first seeks to minimize this risk and then attempts to minimize the cost of the measures in the hardening set.

The primary advantage to using a genetic algorithm for analysis are that the direction of the search can be easily changed by altering the nature of the chromosome or the fitness function. For example, if one is just concerned with finding a set of patches to apply, the chromosome can be redefined as just the set of hardening measures corresponding to patches. The same genetic algorithm described above will still work even with this redefinition. EVA's flexibility in analysis comes from this flexibility that genetic algorithms provides.

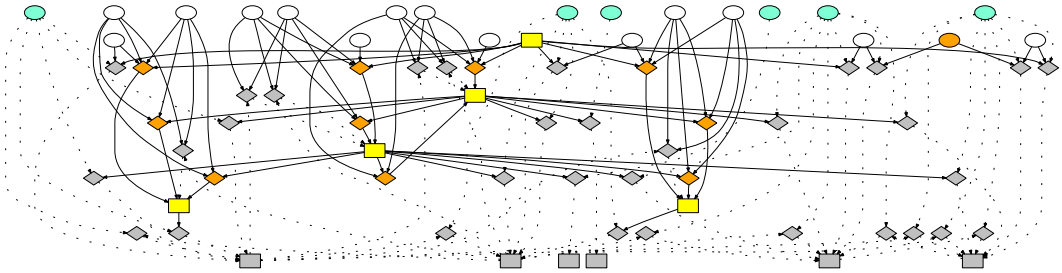
Another advantage to genetic algorithm is many solutions are evaluated in parallel. EVA keeps a record of the best solutions across all generations. Each of these solutions is unique. Currently the ten best solutions are saved, but this is a tunable parameter. When the maximum number of generations has been reached, EVA outputs all of these saved best solutions, ranked by their fitness. The administrator can then choose amongst the solutions. This is particularly useful when multiple solutions with identical fitness exist. The genetic algorithm cannot distinguish between them since their fitness is the same, but a human may have a preference for one solution over another. This is also useful to fine-tune the policy model, described below, to obtain better solutions if the first analysis was not satisfactory to the administrator. By reviewing the saved best solutions, the administrator can see if one hardening measure is being excessively preferred, which could indicate that its cost or benefit needs to be modified.

5.1 Policy Model

The policy model is designed to give the administrator great flexibility in overriding the default behavior of the analysis. The administrator can override the security provided by each class of hardening measures. This would affect how the risk metric is calculated for each goal node. The administrator can also override the cost



(a) Unpatched



(b) Patched

Figure 4: The attack graph and patched attack graph for the malicious student scenario. Since all students are allowed to log in as a user on the lab machines, the analysis cannot disable the user privilege nodes in the patched graph.

of hardening measures. This can be done for a specific hardening measure or a group of hardening measures. The cost can also be changed on different machines.

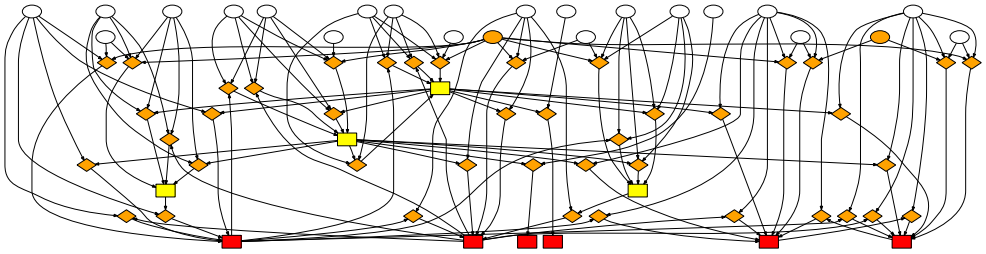
For patches, the policy model allows an administrator to specify an abstract vulnerability class from the abstract exploit templates, a machine name template and the new cost. The abstract vulnerability class corresponds to a class of patches. The machine name template can be an actual machine name, a cluster name or a partial name which will match all machine and cluster names containing that name. The administrator can specify just the vulnerability class or just the machine name template if desired. The most specific cost is used when there is overlap between multiple policies. For example, an administrator can set the cost of a "privilege escalation" class patch to 5 on all machines with one policy rule, but say that the cost of the "privilege escalation" class is only 3 on $host_4$ with another rule. The second rule would be used for $host_4$.

For firewall rules, the policy model allows the cost to be set based on the source of the packet, the destination of the packet and the abstract destination port from the abstract exploit templates. As with patches, the source and destination machine names can be an actual machine name, a cluster name or a partial name. The destination port can be one of the abstract port names or the keyword "all". Similar to the patch policy rules, not all fields need to be specified. If two rules overlap, again the most specific rule will be used. IDS sensor placement has all the fields that firewall rules have and adds a field for the abstract exploit class. The abstract exploit class field allows one to say it is cheaper or more expensive to monitor for certain types of exploits.

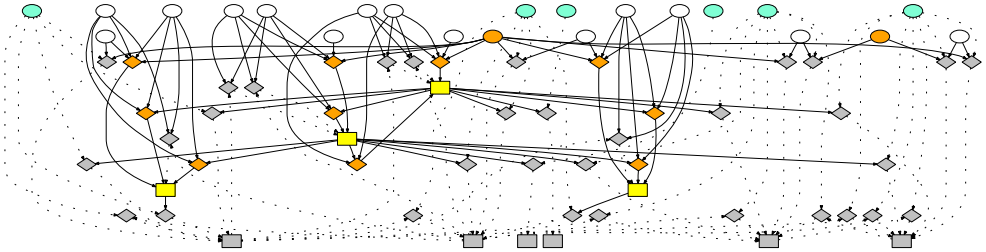
Policy rules can be set for each mode of analysis. Only the rules for the current mode of analysis will be considered. For any hardening measure not covered under a policy rule, the default cost is used. The administrator may alter these default costs for each hardening measure class as well. Default costs can also be altered based on not only the class, but also the mode of analysis.

5.2 Modes of Analysis

The genetic algorithm is adaptable to many modes of analysis. Besides finding a set of hardening measures, it can also be used for strategic planning, network design, forensic evaluation and IDS monitoring. This is done by changing the costs and priorities of each hardening measure (thus altering the fitness of a chromosome), by redefining the chromosome to only consider a subset of hardening measures or by altering the input to the attack graph generator.



(a) Unpatched



(b) Patched

Figure 5: These graphs are for the scenario where a user has a compromised laptop and plugs it in to the instructor's station in a lab. The attacker cannot be prevented from obtaining user privileges since an easily-guessed login is used for student access to the lab machines. The policy prevents this login from being disabled.

For strategic planning, the desired task is to evaluate how the network would respond to unknown risks by performing "what if" scenarios. Essentially, an administrator adds vulnerabilities to the vulnerabilities list file that have not actually been detected in the network and/or alters the connectivity of the network. For example, an administrator would ask "what if machine x has a remote to root vulnerability?" The "what if" scenarios are particularly useful to model vulnerabilities that a vulnerability scanner can not easily find. For example, Nessus cannot detect a client-side browser vulnerability, but this is becoming a common method used to compromise a machine. If the administrator does not have a client-based vulnerability analyzer, he can still model client-side attacks by performing a "what if" scenario. The tool computes the attack graph for the given scenario. The administrator can analyze the resulting attack graph in any of the other supported modes.

With network design, the administrator wants to create a network that is resistant to attack. There are two ways attack graphs can be used to support network design. The simplest method is to have the administrator design several potential networks as input to the strategic planning mode. The tool would then calculate an attack graph for each network and its associated risk metric. The results could then be displayed to the administrator so she can choose the design which has the lowest metric and which best suits the requirements of the installation.

A more interesting approach to network design analysis, and an approach unique to EVA, is to give a prototype network design to the tool and have the tool automatically reconfigure the network to minimize risk. The genetic algorithm in this mode does not consider patches as a possible hardening measure. Instead, it focuses on firewall rules, which could also be interpreted as routing rules, and IDS sensor placement. The fitness function still seeks to minimize the risk of the network. The costs are policy-driven, using the policy rules for network design. The set of firewall rules and IDS sensors that minimizes the risk and minimizes the cost is favored by the algorithm. It outputs several potential network designs that follow this desired outcome.

For forensic evaluation, the current evidence is given as input. This evidence can consist of known resources the attacker has achieved, which corresponds to nodes in the attack graph, or IDS alerts about attacks seen, which corresponds to edges in the attack graph. All evidence that corresponds to the attack graph of the network is highlighted and treated as the initial states of a subgraph of the attack graph. Any other nodes reachable by these states could be other resources the attacker could have compromised. The IDS monitoring mode works simi-

larly, but with current IDS alerts. While it has not been implemented yet, theoretically one could feed the output of the IDS monitoring mode to an intrusion response system. It could then use the knowledge of resources at risk to add further protection measures for those resources. This could prevent the attacker from compromising those resources.

6 Experimental Results

The Computer Science instructional network, as shown in Figure 1, was profiled as the input network to this tool. The network consists of a server zone located outside the firewall and a NAT zone for all the instructional labs. The server zone contains five servers: two Debian Linux servers, one Solaris 8 server, one Solaris 7 server and one Digital Unix server. The instructional lab machines are all identical within a single lab room. There are several prototype lab machines that the administrator clones out to all the machines in a particular room. These prototypes are an Ubuntu Linux image for the general access labs (51 machines), an Ubuntu Linux image for the programming lab (36 machines), a Windows XP image for the hardware labs (24 machines) and an Ubuntu Linux image for the advanced computation lab (30 machines). In total, there are 141 lab machines in the NAT zone and 5 machines in the server zone.

The clustering Perl script derived four clusters based on the vulnerabilities present on the machines and the connectivity allowed by the machines. The first cluster consisted of the servers. The second cluster corresponded to the general access labs. The third cluster corresponded to the programming lab. The fourth cluster contained both the hardware and advanced computation labs since they had identical abstracted vulnerabilities. Even though the actual vulnerabilities differed, the abstracted vulnerabilities are what matters for purposes of clustering. The process of clustering the network took 0.25 seconds on a Xeon quad core 2.33GHz system.

Three "what if" scenarios were also generated for the network. The first scenario assumes that a student in the general access lab is using a version of Firefox with an exploitable vulnerability that would give a malicious website the same privileges on the machine as the student. It is then assumed the student visits such a website, giving the attacker user privileges on that machine. The attacker model states that the attacker's malicious website would place a bot on that machine which would then attempt to compromise other machines and would "call home" to the attacker, thereby allowing the attacker to communicate with the machine even though it is in a NAT.

The second scenario assumes a student has decided to compromise the network. This is a variation of the insider problem. Since the student already has user privileges on all lab machines and several servers, his goal is to escalate his privileges to root on one or more machines. The third scenario assumes an instructor has brought a compromised laptop on to campus. All lab rooms have an Ethernet jack at the instructor station where the instructor can plug in a laptop. There are no restrictions on the connectivity of these jacks. Therefore, once plugged in, they have full access to the LAN containing all the lab machines. Again, this scenario assumes the compromised laptop can "call home" to the attacker so the attacker can have direct access into the NAT zone via the laptop.

All three scenarios and the base configuration of the network were given as input to the attack graph generator. The attack graph for the base scenario showed that two of the servers could be compromised via "remote to root" vulnerabilities. These were two old servers approaching end-of-life which had not been maintained recently. The attack graph for the Firefox vulnerability scenario showed that once the attacker had a foothold into the NAT zone, he was able to get user on all lab machines via the "student" account, which is the account all students use to log in to the lab machines locally. The cluster containing the hardware and advanced lab had a "remote to root" vulnerability that the attacker was able to exploit to get root privileges on those machines. The programming lab had a "privilege escalation" vulnerability that allowed the attacker to elevate from user to root on those machines.

The attack graph for the malicious student showed a similar course of action. The student is able to escalate from user to root on the programming lab machines. The student is also able to exploit the "remote to root" vulnerability on the hardware and advanced lab machines. Likewise, the attack graph for the rogue laptop also showed these compromise routes once the laptop had been plugged into the NAT zone. The generation of each of these attack graphs took 0.5 seconds on the aforementioned quad core Xeon machine.

Each attack graph was then given to the hardening mode of EVA. The goal given to the analysis was to prevent the attacker from gaining root privileges on any machine. The analysis was further restricted to only considering patches that could be applied, instead of all possible hardening measures. The policy rules applied to the evaluation were that logins could not be turned off to any machine and on the lab machines the "student" account could not be disabled, even though it has a guessable password. A run was made without these policy rules and several of the highly fit solutions proposed by the genetic algorithm did indeed suggest these courses of action. When the policy rules were applied, none of the highly fit solutions contained these courses of action.

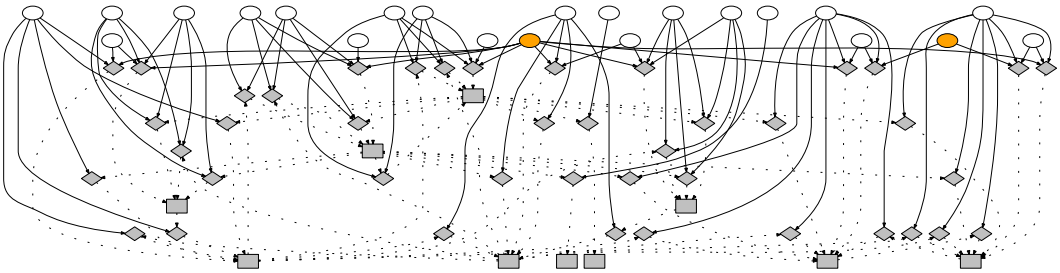


Figure 6: This is the attack graph for the rogue laptop scenario after redesigning the network to segment off the Ethernet port at the instructor's station. Since the laptop plugged into that port can no longer connect to the lab machines, it is unable to compromise them even though the easily guessed student login from Figure 5 remains.

The genetic algorithm was run several times using different population sizes and different maximum generation limits. Larger population sizes will more frequently generate optimal results, but require more CPU time to complete the analysis. A larger maximum generation limit likewise can increase the optimality of the result, but also takes more CPU time. Part of the testing was determining values for these two parameters that balanced good results against CPU time. In doing so, a "suggested parameters" matrix can be developed for other networks that are similarly sized.

When the base configuration was evaluated, the suggested course of action was to patch two servers which had "remote to root" vulnerabilities. No other courses of action were suggested because the remaining machines are inside the NAT and the attacker did not have a vector into the NAT zone in the base configuration. It took 0.01 seconds to evaluate the base scenario using a population of 50 chromosomes and 50 maximum generations for the population. The original and patched attack graph for the base scenario are shown in Figure 2.

For the Firefox scenario, the suggested course of action was to patch the two servers, as before, and to patch the Firefox vulnerability that gave the attacker a foothold into the NAT zone. Again, the genetic algorithm was run with a population of 50 and 50 maximum generations. It took 0.04 seconds for the genetic algorithm to derive this recommendation. The attack graph and analyzed attack graph for this scenario are shown in Figure 3.

For both the malicious student scenario and the rogue laptop scenario, the suggested course of action was to patch the two servers, patch the privilege escalation vulnerability in the programming lab and patch the remote to root vulnerabilities in the hardware and advanced labs. This limits the attacker to just getting user privileges on the machines via the "student" accounts, since it was not allowed to disable those accounts. Again, with a population of 50 and 50 maximum generations, it took 0.03 seconds for the genetic algorithm to derive these recommendations for each scenario. The attack graphs for the malicious student scenario are shown in Figure 4 and the graphs for the rogue laptop are shown in Figure 5.

6.1 Network Design

The three scenarios were also analyzed using the network design mode. For all scenarios, the most fit solutions only required new firewall or router rules. None of the recommendations included placing an IDS sensor for this data set.

For the Firefox vulnerability scenario, it was assumed that the vulnerability was just in the general access labs. The most fit recommendation stated to block Firefox in the general access labs, since there was no policy rule stating to avoid this action. Since blocking Firefox was considered the cheaper course of action, it was recommended over segmenting the NAT zone. With a population size of 250 and 250 maximum generations, the genetic algorithm was able to find this solution on the majority of its runs. It took on average 1.3 seconds to find this recommendation.

For the malicious student scenario, it was assumed the student just had class in the campus-wide general access lab. The student was assumed to not have physical access to the programming, hardware and advanced computation labs. The most fit recommendation was to segment the general access labs away from the remaining labs. Again, a population of 250 and 250 maximum generations were needed to consistently produce this result. It took an average of 1 second for the algorithm to run.

For the compromised laptop, the most fit design was to segment the laptop Ethernet jack at the instructor's station away from the rest of the labs in the NAT zone. As before, a population of 250 and 250 maximum generations were needed. It took an average of 1.05 seconds to calculate. Figure 6 shows the attack graph after the laptop port has been segmented into a different subnet.

This mode needed a larger population size and a higher maximum generation limit than finding a patch set because there were more possible solutions. The number of

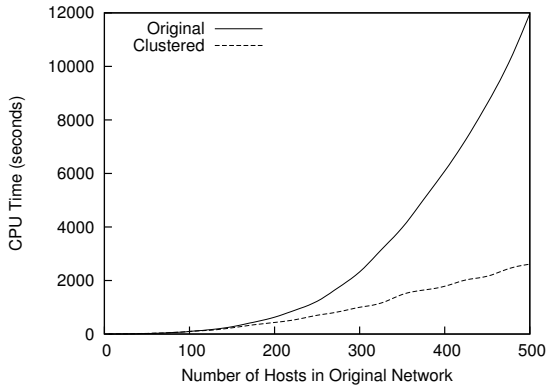


Figure 7: The CPU time for running the attack graph generator and analysis in hardening mode for generated networks with 5 to 500 machines. The original time is for the unclustered machines. The clustered time includes the time it takes to cluster the machines before generating and analyzing the graph.

edges in an attack graph are far greater than the number of nodes in an attack graph since most nodes are highly connected. Determining how to segment the network involves finding the minimal set of edges to cut to disconnect the attacker from the goal nodes, while finding a patch set involves finding a minimum set of nodes to disable. Since there are more edges than nodes, the network design mode has more possible solutions than deriving a set of patches.

6.2 Scalability Testing

The simulated network described in [5] was run through the clustering script, had the attack graphs generated and then was evaluated using the hardening mode in order to test the scalability of this approach. Previously in [5], the tool was tested to a network with 500 unclustered machines. Those same networks were clustered and run again.

For both the unclustered and clustered networks, the proposed hardening measures completely prevented the attacker from getting root privileges on any machine in the network. Figure 7 shows the CPU time of the two methods when the genetic algorithm had a population of 250 and 500 maximum generations. The CPU time is used for this figure since the results in [5] did not use a multi-threaded form of the genetic algorithm. Comparing the CPU time allows the clustering results, which do use the multi-threaded algorithm, to be meaningfully compared to the single-threaded algorithm. It is clear that with clustering, it took far less time to derive the hardening set.

Again, the tool was run with multiple values for the population size and maximum generations. This allowed the "suggested parameters" matrix to be filled with information from larger networks than the Computer Science instructional lab network. As expected, the smaller networks needed only small values for these two parameters. The largest network tested, which contained 2500 unclustered nodes and 337 clustered machines, needed a population size of 500 and a maximum generations of 500 to determine a set of patches. It took an hour and a half on the Xeon quad core 2.33GHz system to analyze this graph due to the complexity of the graph and the large genetic algorithm parameters needed to produce optimal results.

7 Future Work

There are still several areas of improvement for this tool. The first area of improvement is the gathering of input data for the tool. Currently, the firewall and routing rules have to be imported by hand. The next improvement will be to automatically import firewall rules using tools that can extract firewall rules from the network. Another area of input automation is the Nessus plugin ID to abstract vulnerability mapping. A student is currently working on a evolutionary technique to scan the plugin description and classify the plugin based on the keywords in the description. If this works, it should greatly reduce the maintenance needed for the abstraction mappings. Of course, another area for input improvement is to support other vulnerability scanners besides Nessus. This is also planned for the tool.

The second area of improvement is the attacker model. Currently, only one attacker is assumed. If one wishes to model multiple attackers, one needs to run several scenarios, similar to what was described in the results section. A future improvement is to allow multiple attacker models for a single attack graph. This will require marking the nodes to identify which attackers have gained that node and altering the genetic algorithm to pay mind to this node marking.

Another area of improvement is the visualization of the attack graphs. While DOT [3] is nice for small networks, it does not visualize large networks well. A better visualization technique would allow an administrator to "drill down" into the graph to see more specific details or "zoom out" to see more general details.

On the analysis side, one desired area of improvement is to integrate the IDS correlation mode with an intrusion response system to see if it would be feasible to run the analysis in real-time and also if doing so would stop an attacker before they compromised resources. This would be a very powerful extension to the tool.

8 Acknowledgements

I would like to thank the undergraduate students who have worked on this project for their hard work. Jonathan Berling was instrumental in translating the Nessus reports into the appropriate format for attack graph generation and in assisting with the creation of the scenarios that were presented in this paper. Fred McHale and John Millikin played a key role in setting up the isolated network that was used to test the scalability of EVA. I'd also like to thank the Computer Science network administrator, Steve Garcia, and his student assistant, Nick Toothman, for their help in scanning and modeling the Computer Science instructional network.

References

[1] CLIPS: A Tool for Building Expert Systems. [Online] http://clipsrules.sourceforge.net/.

[2] AMMANN, P., WIJESEKARA, D., AND KAUSHIK, S. Scalable, Graph-Based Network Vulnerability Analysis. In *CCS02: 9th ACM Conference on Computer and Communication Security* (Washington, DC, November 2002), ACM, pp. 217 – 224.

[3] AT&T RESEARCH. Graphviz - Open Source Graph Drawing Software. [Online] http://www.graphviz.org/, April 2006. Version 2.8.

[4] DANFORTH, M. *Models for Threat Assessment in Networks*. PhD thesis, University of Califonia, Davis, Davis, CA, USA, June 2006.

[5] DANFORTH, M. Scalable Patch Management using Evolutionary Analysis of Attack Graphs. In *Proceedings of the 7th International Conference on Machine Learning and Applications* (San Diego, CA, USA, December 2008), pp. 300–307.

[6] DAWKINS, J., CAMPBELL, C., AND HALE, J. Modeling Network Attacks: Extending the Attack Tree Paradigm. In *Proceedings of the Workshop on Statistical and Machine Learning Techniques in Computer Intrusion Detection* (June 2002).

[7] DEWRI, R., POOLSAPPASIT, N., RAY, I., AND WHITLEY, D. Optimal security hardening using multi-objective optimization on attack tree models of networks. In *CCS '07: Proceedings of the 14th ACM conference on Computer and Communications Security* (New York, NY, USA, 2007), ACM, pp. 204–213.

[8] FRIEDMAN-HILL, E. JESS: Java Expert System Shell. [Online] http://www.jessrules.com. Version 6.1p6.

[9] INGOLS, K., LIPPMANN, R., AND PIWOWARSKI, K. Practical Attack Graph Generation for Network Defense. In *Proceedings of the 22nd Annual Computer Security Applications Conference* (Miami, FL, USA, December 2006), pp. 121–130.

[10] JAJODIA, S., NOEL, S., AND O'BERRY, B. *Managing Cyber Threats: Issues, Approaches and Challenges*. Kluwer Academic Publisher, 2003, ch. Topological Analysis of Network Attack Vulnerability.

[11] JHA, S., SHEYNER, O., AND WING, J. Two Formal Analyses of Attack Graphs. In *IEEE Computer Security Foundations Workshop* (Cape Brenton, Nova Scotia, Canada, June 2002), pp. 49–63.

[12] J.TEMPLETON, S., AND LEVITT, K. A Require/Provides Model for Computer Attacks. In *Proceedings of the New Security Paradigms Workshop* (Cork Island, September 2000).

[13] MIT PRESS RELEASE. MIT Lincoln Laboratory software aims to thwart cyber hackers. [Online] http://web.mit.edu/newsoffice/2008/security-0827.html, August 2008.

[14] NOEL, S., JAJODIA, S., O'BERRY, B., AND JACOBS, M. Efficient Minimum-Cost Network Hardening Via Exploit Dependency Graphs. In *Proceedings of the 19th Annual Computer Security Applications Conference* (Las Vegas, NV, USA, December 2003).

[15] PHILLIPS, C., AND SWILER, L. A Graph-Based System for Network-Vulnerability Analysis. In *Proceedings of the New Security Paradigms Workshop* (Charlottesville, VA, 1998).

[16] RITCHEY, R. W., AND AMMANN, P. Using Model Checking to Analyze Network Vulnerabilities. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy* (Oakland, CA, May 2000), pp. 156 – 165.

[17] SHEYNER, O. *Scenario Graphs and Attack Graphs*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, April 2004.

[18] SHEYNER, O., HAINES, J., JHA, S., LIPPMANN, R., AND WING, J. Automated Generation and Analysis of Attack Graphs. In *Proceedings of the IEEE Symposium on Security and Privacy* (May 2002), pp. 254 – 265.

[19] SWILER, L., PHILLIPS, C., ELLIS, D., AND CHAKERIAN, S. Computer-Attack Graph Generation Tool. In *Proceedings of the DARPA Information Survivability Conference and Exposition II* (June 2001).

[20] TENABLE NETWORK SECURITY. Nessus. [Online] http://www.nessus.org/.

[21] WILLIAMS, L., LIPPMANN, R., AND INGOLS, K. GARNET: A Graphical Attack Graph and Reachability Network Evaluation Tool. In *Proceedings of the 5th International Workshop on Visualization for Computer Security* (Cambridge, MA, USA, September 2008), pp. 44–59.

An Analysis of Network Configuration Artifacts

David Plonka and Andres Jaan Tack
University of Wisconsin-Madison

Abstract

Computer networks and the Internet have become necessary tools in many daily activities; as such, they share the expectation to be "always on" and highly available. Throughout a decades-long evolution of increasing reliance, campus/enterprise networks and Wide-Area Networks (WANs) have been engineered and maintained by an increasingly large set of skilled practitioners, *i.e.,* network operators or engineers. While strikingly similar to the evolution of software and software development by programmers and software engineers, there has not been similar attention to the discipline of network operations as there has to that of software engineering.

In this work, we analyze the deployment and operation of two large networks over a period of five to ten years. Our *analogy-based* approach is to apply software source code artifact analysis techniques to network device configurations. Specifically, we analyze the repositories of router and switch configurations of both a large campus and a service-provider network; these repositories store the actions of hundreds of practitioners maintaining thousands of pieces of equipment over more than ten years time. Our results expose the evolution of these networks both longitudinally in time and by network device types and topological roles. We reverse-engineer operators' work behavior in terms of how they use version control tools, how they change network device configurations, and how long their changes last in a production network. Lastly, we evaluate our proposed analogy between software engineering and network operations, *i.e.,* that network operators are programmers, by comparing and contrasting the analysis of software development to that of modern network operations.

1 Introduction

The evolution of network engineering and operation has brought it to the point of being the respected profession of increasingly skilled practitioners. This evolution has brought with it tools and techniques which make the administration of large networks feasible. Networking practitioners in these large networks use integrated development environments (IDEs) to guide and control their changes and they use source code management tools to communicate with each other and record a history of their work. Networks, like software projects, have "bugs," *i.e.,* configurations that have negative effects on the system. Also like software projects, networks are subject to the culture of its governing practitioners.

An artifact is defined as "any object created by humans, especially one remaining from a particular period." The software engineering profession has coined the term, "software artifacts," to mean specifically any such object produced by human being *during the course of software development*. These artifacts include code, bug databases, communications, design documents, and revision histories by Source Code Management (SCM) and Version Control Systems (VCS). Following from this, we define *network artifacts* as anything produced by network practitioners in the course of their practice. Matching the world of software, these include device configurations (code), trouble tickets ("bug" reports), communications, design documents, and configuration change histories.

We find the similarity between the software and networking professions compelling. It suggests to us that the two professions may be closely related. However, whereas software has received a great deal of attention from the research community with respect to artifacts and practitioner workflow, the artifacts of network practitioners have gone woefully unstudied. We hypothesize that, just as the analysis of software artifacts has made an impact in the software domain, a similar analysis would be prudent in the networking domain.

We herein propose an analogy-based approach to the analysis of network artifacts, concentrating specifically on the VCS repositories of two long-standing networks as case studies. Our examination makes use of existing

tools designed for software version histories as well as our own longitudinal static analysis of device configurations. While we test our hypothesis, we point out that our approach is unprecedented in the networking community. Therefore, while we might expect some natural similarities, we must be prepared to witness patterns in network practice which do not have obvious counterparts in software development. It is discovering the extent of their similarity that is our motivation.

In this paper, we use the following set of terms to refer to elements of network configuration management repositories and network configurations (similarly to source code management and software source code):

practitioner regardless of domain, the actor or author that is responsible for a configuration change. In the network domain, the practitioner is a network operator or engineer; in the software development domain, this is the programmer or software engineer.

revision a file revision expressing a change to a *single* device configuration. This is the smallest representable change in the systems under study and typically is the work of one authoring practitioner.

commit a set of one or more supposedly related revisions, submitted for storage in a repository by a practitioner. In some prior work, the commit is known as a transaction; we use the CVS command name, `commit`, instead. (In this work we used a window of six hours to coalesce related revisions with `cvs2cl`.)

module a component of the system under study. In the networks we study, the modules are either collections of devices by similar topological role (*e.g.,* core, distribution, access) or by device type (*e.g.,* router, switch, firewall, uninterruptable power supply). In software development a module is typically is a sub-directory containing a subsystem or a class of components, such as header files or library functions.

stanza a line, set of adjacent related lines, or a paragraph of configuration with a common purpose. For instance, a single `interface` or `access-list` definition in Cisco's Internet Operating System (IOS) configuration language. (See Listing 1 for a sample IOS configuration fragment.)

LOC lines of configuration. Network devices are typically configured using a vendor-specific declarative language. This metric is roughly comparable to lines of code in more general programming languages.

The rest of this paper is organized as follows. In Section 2 we introduce the two networks that we study. We subsequently present, in Section 3, the existing tools that we applied to our task. We describe the preparation of the network configuration data in Section 4 and point out some of the similarities and differences between software development and network operations. In Section 5, we first present the results of processing this repository essentially as if it contained software source code. Following those results, we introduce two network-specific analyses and results: *(i)* revision lifetimes and *(ii)* stanza-based activity in Subsections 5.5 and 5.6, respectively. Section 6 reports on our expert interview-based validation of our analyses. Lastly, we report related work in Section 7, propose future work in Section 8, and conclude.

2 Networks Under Study

We studied two large networks: a *campus* network and a *service-provider* network.

Table 1 summarizes the characteristics of the two networks under study.

2.1 Campus Network

The campus network under study is a very large network, with approximately 90,000 ethernet access ports and pervasive wireless ethernet access in many campus buildings. In Table 1 note that the number of operators for the campus network is very high, 343 in total. This is due to the fact that the access layer of this network is partially administered by "authorized agents" employed in "end user" departments throughout the campus that use a sort of a network IDE with a web interface to perform changes, rather than a command-line interface as the super-users often use. (AANTS [16] is one example of such a network IDE.) Of the 343 campus operators, 64 of them are network "super users," *i.e.,* the most privileged operators (with similar responsibilities to the 31 operators of the service-provider network). In summary, the campus network is a large IP and ethernet network, with a 3-tiered layout: a set of core and distribution layer routers and switches providing redundant paths to a very large set of ethernet access layer switches.

2.2 Service-Provider Network

The service-provider network is significantly different from the campus network. It is a mostly router-based Wide-Area Network (WAN), with approximately 500 customer sites in nearly as many cities and municipalities. In Table 1, we see that it has been continually operated for more than ten years under the SCM system;

Network	Period (years)	Operators (super users)	Files	Revisions	Lines of Code
Campus	5+	343 (64)	3,839	128,394	2,898,362
Service Provider	10+	31 (31)	519	41,787	163,882

Table 1: Network Characteristics.

actually, the network was created in the late 1980s, and thus has been operated for nearly 20 years in total. We also see that there are many fewer operators, and devices (files) than the campus network. This is to be expected though, given that it contains almost no access layer equipment; the customers of this service-provider operate their own ethernet Local-Area Networks (LANs) and thus access devices are not part of the service-provider network under study.

3 Tools

As mentioned above, our goal is to utilize existing tools to form a database from our repository of RCS files for the Campus and Service-Provider networks. To this end, we surveyed and experimented with many freely-available tools, both from the research and the open source software developer communities. In general, the former seemed more applicable to our research, however the latter were more easily available and functional in that they were often still currently maintained. For instance, we initially intended to use Bloof [8] because it was feature-rich and extensible, but we found it unsatisfactory in that has not been maintained in years, would not build in our modern development environments, and was also lacking set-up documentation. Since most tools were introduced for use with the popular CVS source code management system, it was convenient that we were able to directly convert our two networks' directories of RCS files to modules within a CVS repository. (CVS actually uses RCS underneath.)

In this study we used the following existing tools to analyze both the campus and service-provider network repositories:

StatCvs-XML StatCvs-XML [3] is a statistics tool for CVS repositories that generates a hierarchy of HTML documents and images from CVS log files. It conveniently supplies a web presentation of both longitudinal and summary statistics.

cvs2cl cvs2cl [1] is a tool of singular purpose: it converts a cvs log to a more concise "`ChangeLog`" file. This is useful to us primarily because it implements the sliding-window algorithm described in German and Mockus' work [9], that coalesces individual file revisions into the author's commit transactions.

From the tool selection process, we've learned that there are *a lot* of tools available but many, while perhaps useful to practitioners, do not expose enough of the details (e.g., they only produce bit-mapped graphs rather than tabular numeric data) to facilitate new analyses.

4 Data Preparation and Transformation

In this work, we report on two case studies each involving the analysis of a repository of configuration files for the devices in a large network. Combined, the data comprises over four thousand files, maintained over approximately ten years, by hundreds of authors. Furthermore, the data was managed in two custom network configuration management systems written in 1997; these systems were similar, and both stored device configurations in files such as that shown in Listing 1, using the legacy file revision control system, RCS. Our analyses, however, expect the data to be in a more modern form. Consequently, perhaps it is not surprising that the raw data needed to be pre-processed, and then transformed. Here we describe the ways in which the network configuration data was prepared for our analogy-based analysis as if it were source code for large software systems.

4.1 Converting From RCS to CVS

Most of converting an RCS-based repository to CVS is straightforward because CVS is based on RCS. We simply created a directory structure of *modules* and move the RCS files into that structure. We chose to use modules which represented the position of each device in the hierarchical topology of a network, *e.g.,* core, distribution, or access layers.

One limitation of our conversion to CVS is that, because RCS does not record when a file has been removed, our CVS repository does not contain file deletions information, so network device removal is not exposed by our analysis. While there are some creative proposals for how this limitation might be addressed (such as using the final revision date as an approximate removal date), we chose to simply not report on any devices whose configurations were ever removed in the years studied. Overall

```
version 12.2
no service pad
service timestamps debug datetime localtime
service timestamps log datetime localtime
service password-encryption
!
hostname s-bldg-5-2-access
!
interface FastEthernet1/0/1
   description sample 100Mbps ethernet interface
   switchport access vlan 42
   switchport mode access
   ...
!
ip access-list extended nodhcpserver
   remark Id: ndhcp.acl,v 1.2 2005-05-20 11:26:03 ashley Exp
   deny   udp any eq bootps any
   permit ip any any
!
access-list 5 permit 192.2.0.1
access-list 5 remark Allow foo, bar, and baz servers
access-list 5 permit 192.2.0.10
access-list 5 permit 192.2.0.11
!
end
```

Listing 1: A representative example of IOS configuration code. Most multi-line stanzas types are separated by exclamation points.

it is relatively uncommon to remove devices completely; more often they are replaced, but keep the same device and file name, so are represented accurately.

4.2 Cleaning the Data

In the course of our analysis work, we discovered a few interesting features of the data itself. Some of these (including some non-printable characters) required manual attention to permit a clean analysis. Others appeared as systemic properties of the network revision control system, and deserve attention as they would have appeared as quite distracting anomalies in visualizations of the network history.

For some devices, we discovered revisions where the change committed removed every line of the configuration. These revisions, then, were immediately replaced by whole files (as they were before the removal). We identified the source of this problem as an intermittent failure of the network devices themselves; these failures were not handled sensibly by the network configuration management systems. Although there were a relatively small number of these "empty" revisions (111 in campus and 21 in service-provider), they needed to be removed so that the subsequent revisions would not have all the configuration lines erroneously attributed to a single author. We cleaned these sources with the heuristic that any revision removing 90% or more of the configuration lines, based on the most lines that had ever been observed prior, should be ignored. After manual inspectection of just that subset of candidates, we found that this heuristic yielded zero false positives and we removed all the errant revisions.

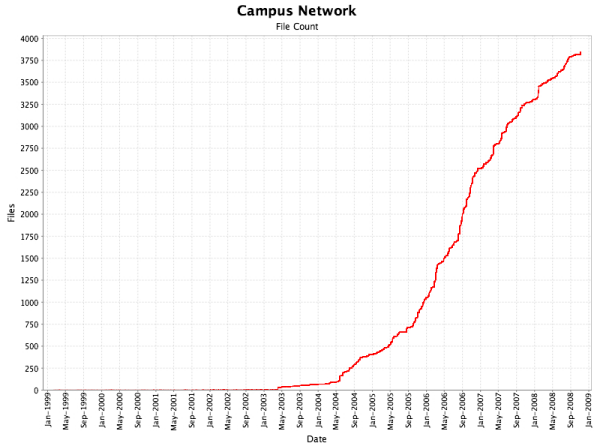
Note that the presence of these empty revisions is a side effect of one major difference between how SCM is done in network operations versus software development. In software development, especially at a large scale, there are many developers, perhaps in many remote locations, that periodically *push* their changed files back to a central repository, from which software releases are subsequently built. By contrast, in network operations, the operators typically operate the SCM system from one central server and they *pull* the configuration file content from the devices' persistent storage (such as non-volatile RAM) back to that central repository. While this *push* versus *pull* model is dramatically different, it has only limited effects on the analysis results. That said, it is worth remembering that networks typically do not have full "development" environments (as in software); the network configuration changes pulled back from devices in the network are *immediately in production*, if they weren't in production already before the revision was committed. (By contrast, software changes typically don't affect a production system until after a software release.)

4.3 Authors and Author Groups

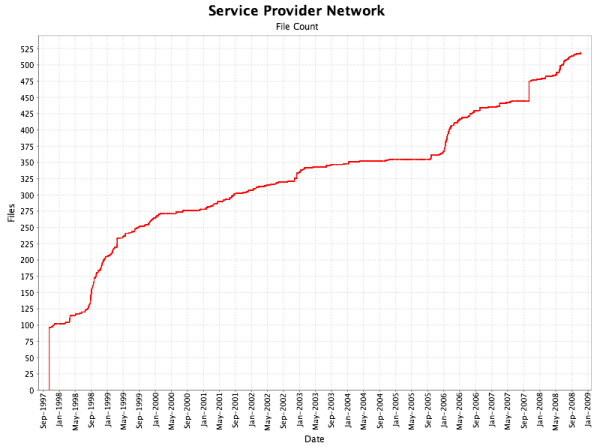
The campus network had very many active operators at 343 in total. Rather than deal with this overwhelming number of authors for visualizations, a portion of our analyses report on groups of operators rather than individuals. The task of translating the practitioner names to their corresponding group was non-trivial because, in ten years, some practitioners had left their jobs, changed to different groups, or even changed names. However, we were able to accurately assign practitioners by using a revision history of their group assignments, kept as described in [12], combined with expert knowledge of the operator employees by other employees that had remained for the duration. Manual effort was also required to combine multiple author (account) names that were really the same practitioner.

5 Analysis and Results

In this section we present graphical and tabular analysis results and comment on characteristics, prominent features, and anomalies that are either similar or different between the campus and service-provider networks under study. Wherever our results mention user login names or real names, these names have been anonymized.



(a)



(b)

Figure 1: Campus (a) and service-provider (b) file/device count over time. These two networks experienced very different growth rates and changes in rate.

5.1 Network Evolution

First, we present the entire lifetime of each network in time series, *i.e.,* each network's evolution in time. While the active portion of the campus network is approximately only five years, both networks are shown in an approximately ten year time range that allows the plots to be easily compared.

Figure 1 shows the number of devices, such as routers and switches, that existed at each point in time for both the campus and service-provider networks. In the campus graph, Figure 1a, notable elements include the growth rate, and its change over time, nearly reaching 4,000 total devices. The shape of this curve suggests that we've captured the network deployment from its inception and that it has gone through periods of differing growth rates. In the service-provider graph, Figure 1b, the adoption of the configuration management system is marked by a sudden increase in device count. There have been two other prominent increases in new devices, beginning roughly September, 1998 and January, 2006, ultimately reaching more than 500 devices in total. Our expert interview from Section 6 was able to offer an explanation for these events.

In Figure 2 we see a time series plot over that same time as Figure 1, but here we show the evolution of the portions of the topology, *i.e.,* by plotting the total LOC for all devices that serve a particular role in the network. We see in both the campus and service-provider networks, that the periphery (campus access layer and service-provider customer sites) are responsible for the most LOC, and that the peripheral topological layers most contribute to the overall growth in configuration content. This is perhaps to be expected as these devices are the most numerous, connecting approximately 90,000 ethernet ports plus wireless access points in the campus and all the service-provider's customers. Another prominent feature is the addition of management equipment after January 2007, and firewall devices after September 2007. However, It is not clear whether these devices were very quickly deployed or whether they were merely inducted into the configuration management system at this time.

5.2 Activity by Topological Role and Device Type

In Tables 2a and 2b we show how much of each module (collections of devices by their topological role) contributes to activity in terms of commits and LOC, for the campus and service-provider networks, respectively. One points of interest is that more than 75% of the commits are performed within each network's periphery (campus access and wireless, and service-provider's customer sites). However, the LOC per commit is quite different between campus and service-provider. This suggests that campus/enterprise access switches require much less fine-tuning than do site routers in this service-provider WAN. We also see that, in both networks, out-of-band management equipment and firewall services represent a much smaller portion of the work, in terms of commits.

5.3 Author Activity

Figure 3 presents the activity for *every* practitioner that authored revisions in the campus and service-provider networks. Because the number of practitioners involved in the campus network is clearly overwhelming, we present the same campus data in Figure 4 based on the group in which they are employed. Specifically, "net"

Module	Commits	LOC	Added LOC	LOC per Commit
campus/access/	89833 (70.0%)	1912430 (66.0%)	2883860 (68.2%)	21.29
campus/access/wireless/	18164 (14.1%)	601836 (20.8%)	657409 (15.5%)	33.13
campus/dist/	7598 (5.9%)	98921 (3.4%)	143155 (3.4%)	13.02
campus/core/	6022 (4.7%)	47272 (1.6%)	97295 (2.3%)	7.85
campus/firewall/	5557 (4.3%)	120147 (4.1%)	319426 (7.6%)	21.62
campus/mgmt/	1220 (1.0%)	117756 (4.1%)	126903 (3.0%)	96.52

(a)

Module	Commits	LOC	Added LOC	LOC per Commit
isp/dist/site/	31931 (76.4%)	92977 (56.7%)	309604 (55.7%)	2.91
isp/dist/hub/	5203 (12.5%)	28116 (17.2%)	98581 (17.7%)	5.40
isp/border/	3373 (8.1%)	18665 (11.4%)	98985 (17.8%)	5.53
isp/firewall/	445 (1.1%)	12516 (7.6%)	25939 (4.7%)	28.13
isp/mgmt/	835 (2.0%)	11608 (7.1%)	22434 (4.0%)	13.90

(b)

Table 2: Commits by topological role of the device for campus (a) and service-provider (b) networks.

is the network engineers, "contract" represents the contractors, "noc" is the Network Operations Center (NOC) staff, "field" the field service agents, "authorized-agents" are employees in various peripheral campus departments that are authorized to make access layer changes only, and "security" is an IT security group. From this pie chart, we see that the operators responsible for most of the LOC are network engineers proper. Also, the contractors performed a significant amount of similar work.

In Tables 3a and 3b, we show the top ten most active practitioners based on their number of commits. Note also that the LOC per commit is approximately an order of magnitude different between the campus and service-provider network operators. This suggests that the campus, with very many switches rather than routers, is in a higher state of flux and perhaps recently in a deployment mode. In contrast, the service-provider network experiences relatively small changes in terms of LOC per commit, perhaps suggesting that it is largely stable and in a maintenance mode.

5.4 Anomalies

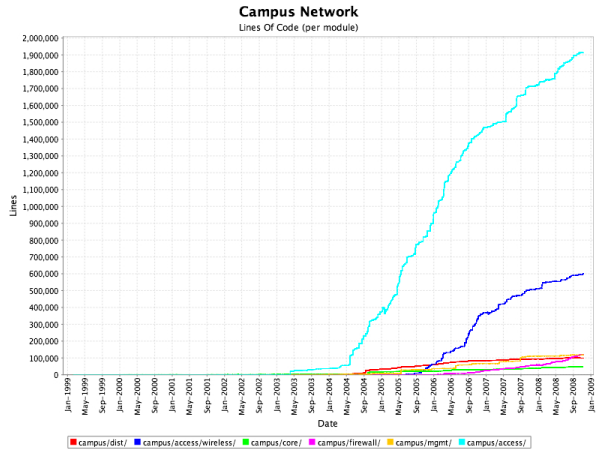
Here, we describe a number of curiosities or anomalies discovered in the networks studied, solely based upon the results presented thus far.

5.4.1 Activity by Campus "system" Author

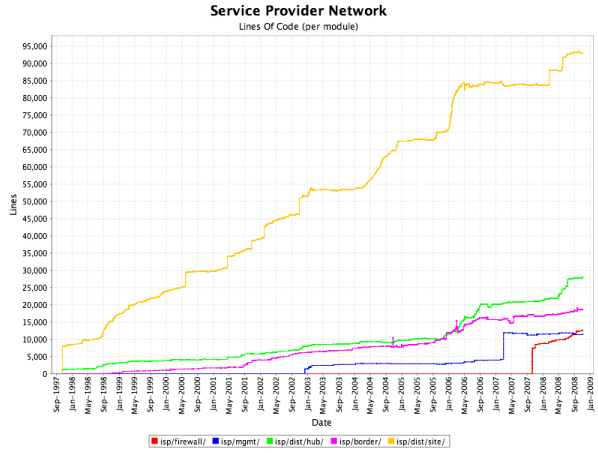
In the campus network, and shown in Table 3a, we can see that one of the "Top 10" most active authors is the software system itself (by the name `system`), rather than a real person/practitioner. This entry is additionally interesting in that overall it has removed more lines that it has added and thus is very different from the real practitioners. Further investigation identified two reasons for this unexpected significant authorship of changes by the SCM system itself: *(1)* some of the operators often do not "follow the rules," *i.e.,* they do not commit their changes in a timely fashion and thus the system sometimes has to commit their changes implicitly just prior to applying a subsequent automated change (so as not to mix unrelated changes together), and *(2)* a few operators have discovered an unintended feature of their automated change system; namely, that they can cause their earlier changes to be committed implicitly to the version repository. This avoids those changes being reported as unfinished in a nightly email report to all operators. Both of these causes demonstrate to how a VCS can produce both efficiencies and inefficiencies in the everyday work flow of network operators. This suggests that the process by which changes are merged into a network configuration version control system can be improved. It is an open question as to whether existing merging techniques from SCM systems will be similarly effective, but there are certainly both syntactic and semantic differences between the network device configuration files in a production network and the source files in software development.

5.4.2 Outstanding Service-Provider Author

Considering the question of which operators perform most commits, we see in Table 3b that both the most commits and most of the LOC are authored a single, seemingly "super human," outstanding author, here named "robert." This suggests that operator involvement

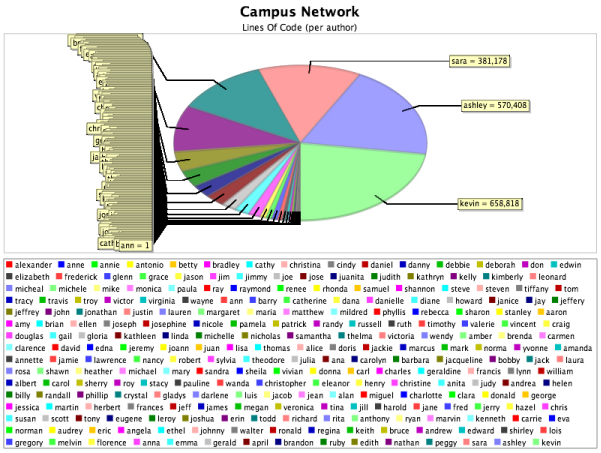


(a)

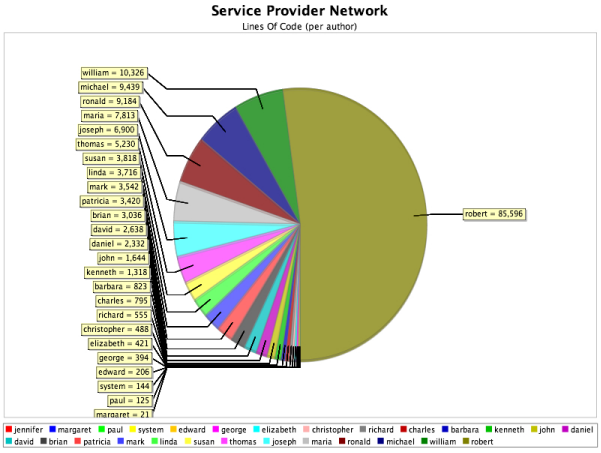


(b)

Figure 2: Campus (a) and service-provider (b) LOC by topological role over time. Most of the LOC are configuration of the periphery of each of these networks, *i.e.*, the campus access layer and service-provider's customer sites.



(a)



(b)

Figure 3: Campus (a) and service-provider (b) LOC per author. In both networks, five authors are responsible for approximately 75% of the LOC.

varies widely amongst networks and amongst individual practitioners with respect to the tasks of introducing devices (*i.e.*, introducing many LOC of their initial configuration) and subsequently managing a network's device configurations.

5.4.3 Common Commit Comments

Tables 4a and Tables 4b show the most common comments provided by the operators in the campus and service-provider networks, respectively. In Table 4a we see that the second most common comment is "asdf," from the home row on a qwerty keyboard, suggesting it's a cavalier refusal to supply a meaningful comment. Further investigation showed that this comment is *always* supplied by only one of the authorized agents using a web interface to perform changes. Unlike with the CLI interface, here the comment is required, and thus the practitioner is forced to supply something. Our hypothesis is that this practitioner likely sees only himself as the "audience" of the comments, and deems it unnecessary to exert effort to carefully explain the changes he commits.

In Table 4b we see that nearly 6% of all log comments are empty. Like the campus "asdf" comment, these empty comments are being supplied by only a small subset of the practitioners, again perhaps ones that don't see, or have never realized, any value from such comments. In the service-provider environment, the "?" comment was occasionally supplied by the outstanding practitioner that performs most of the commits. Further investigation suggests that he is stumbling across changes made

Author	Commits	LOC	Added LOC	LOC per Commit
ashley	16430 (12.8%)	570408 (19.7%)	952945 (22.5%)	34.72
kevin	9296 (7.2%)	658818 (22.7%)	703006 (16.6%)	70.87
system	**8164 (6.4%)**	**-6595 (-0.2%)**	**49117 (1.2%)**	**-0.81**
nathan	5257 (4.1%)	279484 (9.6%)	329512 (7.8%)	53.16
sara	4790 (3.7%)	381178 (13.2%)	410738 (9.7%)	79.58
edith	4755 (3.7%)	122640 (4.2%)	134277 (3.2%)	25.79
brandon	4666 (3.6%)	75641 (2.6%)	91540 (2.2%)	16.21
ruby	4626 (3.6%)	99700 (3.4%)	190530 (4.5%)	21.55
peggy	3958 (3.1%)	345232 (11.9%)	365551 (8.6%)	87.22
emma	3483 (2.7%)	54658 (1.9%)	63449 (1.5%)	15.69

(a) Note that the third most active campus author, "system," is not a practitioner but records automated commit activity by the configuration management system itself.

Author	Commits	LOC	Added LOC	LOC per Commit
robert	**30385 (72.7%)**	**85596 (52.2%)**	**396634 (71.4%)**	**2.82**
michael	1489 (3.6%)	9439 (5.8%)	16443 (3.0%)	6.34
brian	1444 (3.5%)	3036 (1.9%)	15698 (2.8%)	2.10
joseph	1431 (3.4%)	6900 (4.2%)	13688 (2.5%)	4.82
linda	1174 (2.8%)	3716 (2.3%)	13091 (2.4%)	3.17
william	1058 (2.5%)	10326 (6.3%)	14566 (2.6%)	9.76
daniel	673 (1.6%)	2332 (1.4%)	7254 (1.3%)	3.47
john	628 (1.5%)	1644 (1.0%)	4952 (0.9%)	2.62
kenneth	511 (1.2%)	1318 (0.8%)	5461 (1.0%)	2.58
david	459 (1.1%)	2638 (1.6%)	6137 (1.1%)	5.75

(b) Note that the most active service-provider author, "robert," is a single most outstanding operator that performed more than 70% of the commits and was responsible for more than half of the LOC.

Table 3: Commits by author for the (a) campus and (b) service-provider networks. The bold entries are discussed in Sections 5.4.1 and 5.4.2.

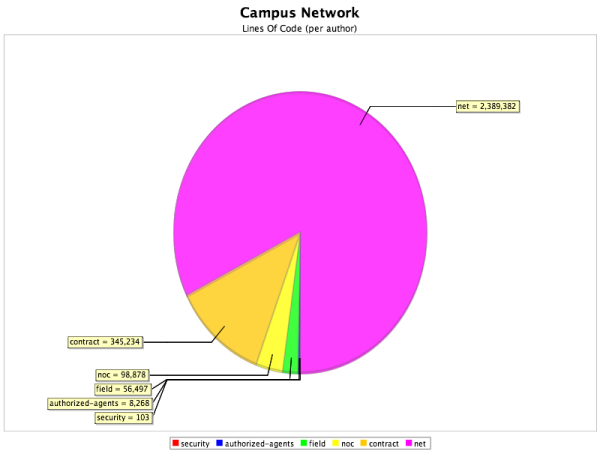


Figure 4: Campus LOC per author group. We note that the "net" (network engineering staff) group is responsible for approximately 80% of the LOC, followed distantly by contractors, field service agents, and authorized agents.

Comment	Frequency
Initial revision	1442 (2.8%)
asdf	**584 (1.1%)**
test	437 (0.9%)
'newer bulk checkin'	411 (0.8%)
change vlan	308 (0.6%)

(a)

Comment	Frequency
***** empty log message *****	**768 (5.9%)**
Initial revision	350 (2.7%)
router swap	117 (0.9%)
config cleanup	107 (0.8%)
?	75 (0.6%)

(b)

Table 4: Top five commit comments for (a) the campus network and (b) the service-provider network. In each of these results, garbage comments indicate operator non conformance and other habits. The bold entries are particularly unexpected and are discussed in Section 5.4.3.

by others, and is essentially using the "?" to say that he's checking in changes performed by someone else, for which he does not readily have an explanation. He thus commits that change, and carries on with his tasks without having to wait for such an explanation.

These anomalous results of system authorship of commits and common log comments both speak to the issue of operator conformance with the system used in these networks. In large part, practitioners appear to use the tools as intended, and with a high degree of compliance. However a subset of the operators seem to find it cumbersome and sometimes find workarounds that make their tasks easier. Such discoveries can effectively guide new tools and features.

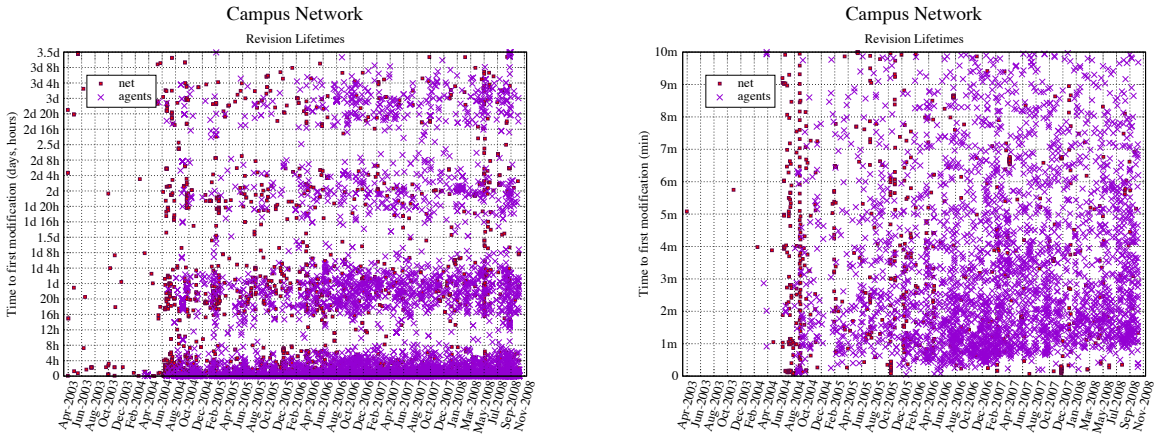
5.5 Revision Lifetimes

In Figure 5, we see a pair of plots demonstrating *revision lifetimes*, or the time from a revision's appearance within a file to the first subsequent revision which affects any of the same lines of configuration. Both plots are for the campus network (the service-provider network does not change often enough for this plot to be valuable). We are particularly interested in short-lived changes, here clustered to the bottom of the graph. Note that this version history is unique in that it always reflects a production environment.

In Figure 5a, we are surprised to see that such short revisions as to occur within a day or two of each other (suggesting a network "bug") are treated only during business days, and very infrequently require overnight attention from network operators even though these revisions are ostensibly part of the production network.

Figure 5b, essentially the same data on a finer time scale, tells its own story about change lifetimes from different contributor groups. The *net* group (squares) represents super-users on the network, whose access is completely unrestricted. This group makes relatively few changes in the ten-minute window shown here. The other group, authorized agents working at all levels of the network infrastructure, composes the vast majority of the plotted points (crosses). These agents make their changes through a web interface (essentially an IDE for the network) which automatically checks in the change as soon as it is applied to the router.

Based on this last observation, we see that we have two different data sets available to us in the revision history for the campus network. For network engineers (the *net* group), we see a traditional software-like history of commits, where the user commits his changes most often after he has observed their effect and deemed them a valuable contribution. From the commits made by *agents*, since they are not privileged to interact with the devices directly, we actually have a richer version history. Their history not only includes those changes which survive in the long term, but also the changes that they make as part of their efforts from one minute to the next. It is, one may consider, an extrapolation of revisions to a perfectly fine granularity of change. Thus, in the recorded history of this network, we find an artifact which is entirely unavailable from any known software project.



(a) (b)

Figure 5: Campus revisions, time to next modification: 3.5 days (a) and 10 minutes (b).

5.6 Activity by Stanza Type

The relatively simple structure of IOS configurations allows some static analyses which consider *stanzas*, rather than lines, as the basic units of change from one revision to the next. Tables 5 describe the results of this analysis. These results can guide the creation of tools to manage the network under inspection: In both cases described here, we confirm that any service built for the configuration of these network devices would be well-advised to cater specifically to the management of `interface` and *global* stanzas.

Stanza Type	Total Revisions	Revisions per Instance
`interface`	471,238	4
`vlan`	25,591	1
global	12,534	4
`logging`	12,390	9
`ip`	12,006	1
`bridge`	4,353	1

(a) Campus network: the ratio of `interface` stanza revisions to *global* stanza revisions is 19:1.

Stanza Type	Total Revisions	Revisions per Instance
`interface`	25,288	4
global	11,737	26
`ip`	8,207	4
`line`	6,146	14
`router`	3,974	4
`policy-map`	2,783	4

(b) Service provider network: the ratio of `interface` stanza revisions to *global* stanza revisions is roughly 2:1.

Table 5: Number of revisions made per each IOS stanza type, for the campus (a) and service provider (b) networks. The *global* meta-stanza included all unindented lines at the top of a file, preceding the appearance of any others in this list.

5.7 Discussion

We close this section with our observation about LOC as a metric for networks rather than software. While we have not yet done analysis of code complexity, early indications suggest that there are a number of reasons that numbers of lines of configuration (LOC) is a poor candidate as a measure of complexity or work. First, the initial versions of our configuration files (source code) contain very many "boiler-plate" lines produced by the network device itself; attributing these lines of code to the operator that introduced the device to the network dramatically exaggerates the volume of the work done by that operator. Secondly, the configuration files are rigidly formatted by the device rather than the operator (programmer), *i.e.,* it is not a free format language. Thus, the vendor-specific network device configuration language, itself, dictates the numbers of lines more so than modern general software programming languages dictate the number of lines of program source code.

6 Validation

This being an initial study of it's kind, to the best of our knowledge, we were left to interview domain experts in network operations to validate our approach. For the campus network, we interviewed the manager to whom most of the super-user operators have reported. For the service-provider network, we interviewed the director.

6.1 Campus Expert Feedback

Here are highlights of the feedback offered by our campus network expert:

- The top authors by LOC agrees with managements knowledge of their respective performance, *i.e.,* these are outstanding practitioners in that they indeed have the most responsibility for network equipment deployment.
- The data points, *e.g.,* commit volume and common comments, would be useful to demonstrate to customer departments that we know how authorized agents use the tools provided.
- The visualizations are useful to show the evolution of the network's architecture over time, *e.g.,* the wireless access deployment and the use of contract labor to do so.
- The author-specific visualizations, such as activity by days of week and times of day would be an interesting addition to existing tools, such as the network IDE provided to the practitioners themselves.

6.2 Service-Provider Expert Feedback

Here are highlights of the feedback offered by our expert on the service-provider network:

- The file count evolution over time clearly shows inflections due to two significant events: *(i)* a $200M influx of funding resulting in membership growth by more than 100 sites and *(ii)* the merging of the service-provider network with a similarly scoped network, resulting in many devices being replaced (to switch from T1 circuits to 10Mbps ethernet).
- One practitioner, a temporary employee, was responsible for an unexpectedly large number proportion of the code. However, this coincides with the person's role, which was to deploy replacement equipment. (Consequently, they were responsible for much of the initial device configuration, thus a large number of lines of configuration.)
- The similarity between network operations [when viewed this way] and software development is striking.
- Common commit comments suggest the need for a new standard operating procedure that would encourage practitioner's to supply meaningful comments; this would also aid analysis.
- Such linear trends over time were not expected. There are some events that had significant costs (such as router replacements by alternate brands) that do show prominently in the time series graphs. (This is akin to, perhaps, changing programming languages in a portion of a software system.)

While clearly a subjective assessment, the feedback from both experts showed the utility of our results, and consequently the value of the analogy-based application of these analyses.

7 Related Work

We are aware of one study in the literature, the recent work of Sung, *et al.* [15], that longitudinally examined network configuration repositories of network devices such as routers and switches. Similarly, our work also examines and reports on the configuration changes in multiple real-world networks over time, examines stanzas by type, and evaluates results by expert interview. However, our work differs in that we apply software development analysis techniques to expose practitioner behaviors and network evolution over time, whereas they apply different data mining techniques to identify correlated configuration changes. More generally, our work is informed by related work in three areas: programming languages, network management, and systems administration.

The Revision Control System (RCS [17]) is the version control sub-system with which the versions of configurations we consider are stored. In [4], Ball, *et al.,* demonstrate some of the uses of the information stored in such VCSs for software source code. Our work applies analysis and visualization techniques to expose characteristics of network management in a similar fashion to that early examination of software development via VCS. In [7], Draheim and Pekack introduced a freely-available tool, Bloof [8]. Tools such as Bloof and cvsanaly2 [2], introduced in work [14] by Robles, et al., could potentially be used similarly to the one we used (StatCvsXML).

In this work, we study repositories of network configurations maintained by the Network Configuration Management System (NetCMS [11]) and AANTS [16]. An alternative technique often used by network operators is to retrieve device configurations using RANCID [13, 10]

and subsequently store them using tools such as CVS. A very recent work [5] by Benson, *et al.,* introduces a code complexity metric for network devices configurations. Their metric uses attributes including Lines of Code and inter-stanza references (within and amongst configuration files) to arrive at a numeric measure of complexity; they subsequently validate their proposed metric by operator interview. In this work, instead, we develop a way to measure programmer effort by revision lifetimes, but have not yet used it to evaluate a complexity metric.

There is a large literature concerning the profession of system administrator and improvement to the processes involved in system configuration. System administrators sometimes similarly use VCSs for their configurations [12] and researchers seek to improve configuration management. For instance, Sun and Couch develop a state-machine model of configuration management in [6].

8 Future Work

While we have completed an analysis of two ostensibly different, large networks, the process and results suggest some directions for future work.

In our consideration of revision lifetimes, we have not considered the author of the subsequent revision. It may be useful to classify or characterize practitioners based upon the lifetimes of the revisions they make. Also, one might consider whether or not practitioners do a revision that modifies the configuration that they introduced in a earlier revision, or whether or not practitioners just as easily (and often) maintain each others configuration fragments.

In this work, we did not much consider how the declarative configuration can be influenced by the revising practitioners intent or style. This because the layout of the configuration is nearly completely dictated by the device operating system. However, there are a subset of stanza types that allow for more variety in the expression of their purpose. For instance, access control lists (ACLs) contain statements that can be ordered by the operator, and multiple orderings and arrangements can have the same effect; some orderings are likely more concise or understandable than others. Therefore, it may be fruitful to consider whether or not some revisions are simply refactorings, like in software development. Further, the identification of cloned configuration fragments amongst devices, as in code clone analysis of software, could identify oft used configuration idioms.

Lastly, the goal of measuring effort in terms of revisions lifetimes was to provide a measurement of complexity. For instance, one might wonder which stanza types are more complex as evidenced by their modification (presumable fixes) in rapid succession. We did not implement nor even propose a complexity metric in this work, but future work could explore this topic, and determine whether or not certain refactorings are more or less complex.

Conclusion

In this paper we presented two techniques: *(i)* an initial application of software development analysis tools to network operations and *(ii)* the beginnings of network operations-specific approach to measuring practitioner effort to guide new tool development. We applied these techniques in case studies of the network configuration repositories of both a large campus network and a service-provider network. By analysis and visualization, we compared and contrasted the two networks, investigating the value of metrics (*e.g.,* LOC) and exposing practitioner behaviors when using SCM and IDE-like tools. Lastly, we evaluated the analogy-based application of software development mining tools to the discipline of network operations by performing expert interviews. This expert feedback suggests the promise of our approach as both a technique to visualize the operation of real networks and as an aid to management and other stakeholders in understanding where operational effort is concentrated in large computer networks.

In closing, we have provided evidence that existing software development analysis techniques are of significant value when applied in the network operations domain. These methods expose practitioner behavior and essentially show that network operators *are* programmers, at least in their use of similar tools. By analogy to software development, this suggests that the study of network operations can effectively inform and direct network management tool development. Our hope is that the resulting improved tools will liberate the network operator from mundane tasks, will reduce mistakes in configuration, and will enable skilled operators to focus their efforts more completely on the goal of continually increasing network reliability.

References

[1] cvs2cl. `http://www.red-bean.com/cvs2cl/`.

[2] cvsanaly2. `http://forge.morfeo-project.org/projects/libresoft-tools/`.

[3] StatCvs-XML. `http://statcvs-xml.berlios.de/`.

[4] BALL, T., MIN KIM, J., PORTER, A. A., AND SIY, H. P. If Your Version Control System Could Talk. In *In ICSE '97 Workshop on Process Modelling and Empirical Studies of Software Engineering* (1997).

[5] BENSON, T., AKELLA, A., AND MALTZ, D. Unraveling the Complexity of Network Management. In *NSDI '09: Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation* (2009).

[6] COUCH, A., AND SUN, Y. On Observed Reproducibility in Network Configuration Management. *Science of Computer Programming 53*, 2 (November 2004), 215–253.

[7] DRAHEIM, D., AND PEKACKI, L. Process-Centric Analytical Processing of Version Control Data. In *IWPSE '03: Proceedings of the 6th International Workshop on Principles of Software Evolution* (Washington, DC, USA, 2003), IEEE Computer Society, p. 131.

[8] DRAHEIM, D., AND PEKACKI, L. The Bloof Project. `http://bloof.sourceforge.net`, 2003.

[9] GERMAN, D., AND MOCKUS, A. Automating the Measurement of Open Source Projects. In *Proceedings of the 3rd Workshop on Open Source Software Engineering* (2003), pp. 63–67.

[10] GOULD, W. Backing up your network with RANCID. `http://www.linux.com/feature/55873`, 2006.

[11] PLONKA, D. NetCMS - Network device Configuration Management System. `http://net.doit.wisc.edu/~plonka/NetCMS/`, 1997.

[12] PLONKA, D. Sys Admin File Revision Control with RCS. *SysAdmin - the Journal for UNIX Systems Administrators* (1998), 8–24.

[13] RANCID - Really Awesome New Cisco Config Differ. `http://www.shrubbery.net/rancid/`.

[14] ROBLES, G., KOCH, S., AND GONZALEZ-BARAHONA, J. Remote Analysis and Measurement of Libre Software Systems by Means of the CVSAnalY Tool. In *Proceedings of the 2nd ICSE Workshop on Remote Analysis and Measurement of Software Systems (RAMSS), Edinburg, Scotland, UK* (2004), pp. 51–55.

[15] SUNG, Y., RAO, S., SEN, S., AND LEGGETT, S. Extracting Network-Wide Correlated Changes from Longitudinal Configuration Data. In *Proceedings of the 10th Passive and Active Measurement Conference (PAM)* (2009), Springer, pp. 58–67.

[16] THOMAS, C., AND PLONKA, D. AANTS: Web-Based Tools for Cooperative Campus Network Administration. In *Proceedings of the Fall 2005 Internet2 Member Meeting, Philadelphia, PA, USA* (2005).

[17] TICHY, W. F. Design, implementation, and evaluation of a Revision Control System. In *ICSE '82: Proceedings of the 6th International Conference on Software Engineering* (Los Alamitos, CA, USA, 1982), IEEE Computer Society Press, pp. 58–67.

Secure Passwords Through Enhanced Hashing

Benjamin Strahs Chuan Yue Haining Wang
Department of Computer Science
The College of William and Mary
Williamsburg, VA 23187, USA
{*bgstra,cyue,hnw*}*@cs.wm.edu*

Abstract

Passwords play a critical role in online authentication. Unfortunately, passwords suffer from two seemingly intractable problems: password cracking and password theft. In this paper, we propose *PasswordAgent*, a new password hashing mechanism that utilizes both a salt repository and a browser plug-in to secure web logins with strong passwords. Password hashing is a technique that allows users to remember simple low-entropy passwords and have them hashed to create high-entropy secure passwords. PasswordAgent generates strong passwords by enhancing the hash function with a large random salt. With the support of a salt repository, it gains a much stronger security guarantee than existing mechanisms. PasswordAgent is less vulnerable to offline attacks, and it provides stronger protection against password theft. Moreover, PasswordAgent offers some usability advantages over existing hash-based mechanisms, while maintaining users' familiar password entry paradigm. We build a prototype of PasswordAgent and conduct usability experiments.

1 Introduction

Passwords remain the most common security method to authenticate or verify a user's online identity [25]. They provide a powerful guard against unauthorized access to systems and data, and are ubiquitously used in various online activities such as shopping, banking, communication, and learning. User authentication via password relies on the *something you know* authentication factor, i.e., you know some secret that no one else does. Although two other authentication factors *something you have* (e.g., hardware token) and *something you are* (e.g., fingerprint) have also been recognized and used in practice, they have not gained a wide acceptance on the Internet, primarily because of their high cost, limited flexibility, and restricted portability. On the contrary, passwords are simple, inexpensive, easy to implement, and convenient to use. Consequently, they occupy the dominant position in online user authentication, and this situation will not change in the foreseeable future.

Despite their prevalence and importance in online authentication, passwords do have two well-known and long-standing problems: weak passwords are easy to crack, and passwords are vulnerable to theft. Password security depends on creating strong passwords and protecting them from being stolen. A strong password should be sufficiently long, random, and hard to discover by crackers. In contrast, a weak password is usually short, common, and easy to guess. Examples of strong passwords include "t3wahSetyeT4" and "Tpftc-its4Utg!"; and examples of weak passwords include "susan123" and "password" [6]. Weak passwords suffer from vulnerability to brute-force and dictionary attacks [29]. The dilemma in a password system is that a user will often choose guessable passwords simply because they are easy to remember [11, 19, 29]. Moreover, no matter how strong they are, passwords are also vulnerable to theft. One of the most significant threats to online users is *phishing* attack [18, 39], which uses social engineering techniques to steal users' personal identity data on spoofed websites [1]. In recent years, this type of identity theft has risen sharply and has cost billion-dollar losses [2]. Other attacks like *shoulder surfing* [33] can also steal user passwords, especially in public places such as cybercafes, airports, and libraries.

As more online services are password-protected, users have to create and memorize an increasing number of passwords. This, combined with the inherent limitation of human memory, forces users to revert back to insecure habits such as choosing simpler passwords, reusing passwords across different websites, or even writing down their passwords [37]. A recent large-scale study of web password usage shows that on average, a user has approximately 6.5 passwords shared across 25 different

websites, and the majority of users choose weak passwords that contain only lower case letters [22].

To address these problems and enhance online password security, a number of techniques have been proposed. For example, *password managers* generate strong passwords and automatically store them in a local database [8, 5]. *Single sign-on* systems allow users to log into many websites using one account, which reduces the number of passwords a user must remember [9]. *Graphical passwords* enable users to click on images to authenticate themselves [26, 17]. However, these solutions all have their own limitations. *Password managers* store passwords on a fixed computer and thus lack mobility; *single sign-on* systems place too much trust on a centralized system and thus are vulnerable to single-point failure [27]; and *graphical passwords*, although proposed as an alternative to traditional text-based passwords, are still hampered by security and usability concerns [16, 34].

A promising approach to obtaining secure online passwords is *password hashing*, in which hashed passwords are sent to remote websites instead of plain-text passwords. Password hashing is very attractive for a few reasons: it is lightweight and convenient to use, increases password strength, and can defend against phishing attacks. This approach has been taken in projects such as the Lucent Personal Web Assistant (LPWA) [23], PwdHash [31], Password Multiplier [24], and Passpet [38]. However, these systems still have security limitations which will be discussed in Section 2. Moreover, password hashing systems, if not carefully designed and implemented, suffer from usability problems that may directly lead to security exposures [14].

In this paper, we present *PasswordAgent*, an automatic password management system with enhanced hashing, which consists of a Salt Repository server and a browser plug-in Agent for securing online passwords. The Salt Repository stores a list of salts for each registered user while the Agent provides the user interface, salt retrieval, and hashing functionality. When a plain-text password needs to be protected for a specific website, the user simply activates the Agent and enters the plain-text password. The Agent automatically concatenates the plain-text password and the website specific salt to deterministically generate the site password via a hash function.

The contribution of PasswordAgent to online password management lies in the following aspects. First, it automatically provides a stronger security guarantee by using randomly generated and securely stored salts. Second, it improves phishing protection by giving users accurate warnings if they attempt to enter protected account information on an unprotected website. Moreover, even if phishers obtain the plain-text passwords by using subtle techniques such as JavaScript attacks or "spoofed password field in Flash" [31], they still cannot access users' accounts because they do not have the salts. Third, as long as the password to the Salt Repository server is not observed by an attacker, PasswordAgent also reduces the risks of shoulder surfing attacks. Finally, a few usability suggestions made in [14] are incorporated into PasswordAgent, providing some usability advantages over existing solutions.

The remainder of this paper is structured as follows. We describe existing password hashing solutions in Section 2. We present the design of PasswordAgent in Section 3 and analyze its security and usability in Section 4. We detail the implementation and usability evaluation of PasswordAgent in Section 5. We discuss the limitations of PasswordAgent in Section 6, and finally we conclude in Section 7.

2 Related Work

In this section, we highlight the contributions of the Lucent Personal Web Assistant (LPWA) [23] and three recent systems: PwdHash [31], Password Multiplier [24], and Passpet [38]. These existing systems exemplify the concept and value of password hashing in online user authentication, and they are most related to our proposed PasswordAgent.

LPWA is an HTTP proxy providing data anonymity services to users. To a user, LPWA generates secure, consistent, and pseudonymous usernames, passwords, and email aliases for different websites based on three inputs: a UserID, a universal password to the proxy, and a destination website address. Using LPWA, users can protect their real identities and weed out junk email based on the recipient email address. LPWA was successful before, but now it has serious limitations. LPWA does not support HTTPS, but the identities that need to be protected the most are those that are transmitted via HTTPS. LPWA also requires users to fully trust the proxy server, which knows all the login credentials to the destination servers, resulting in security and privacy concerns.

PwdHash is a browser extension that transparently creates a different password for each site, improving web password security and defending against phishing attacks. PwdHash addresses a few challenges of implementing password hashing as a secure and transparent extension to web browsers. In particular, PwdHash uses the destination domain name as a salt and sends a hashed password to the remote site. However, PwdHash, as acknowledged by the authors, is vulnerable to two major kinds of attacks. One is a dictionary attack on the hashed passwords. This vulnerability is due to three factors: a phishing site can obtain hashed passwords, PwdHash uses MD5 [30], a very fast hashing algorithm, and the salt is publicly known. The second vulnerability of Pwd-

Hash is its susceptibility to advanced phishing attacks, such as using Flash objects or focus stealing. Flash objects and focus stealing are a form of phishing that allows keyboard strokes to be intercepted before other browser plugins have a chance to handle them.

As a browser extension, Password Multiplier can generate strengthened passwords for an arbitrary number of accounts while requiring the user to memorize only a single short password. It uses the same three inputs as LPWA: a UserID, a master password, and a destination domain name. The key contribution of Password Multiplier is using a strengthened hash function to deterministically generate high-entropy passwords. However, the main problem with Password Multiplier is that all the derived passwords will be known to attackers if the master password is stolen. At present, it is possible for an attacker to steal a master password through a keylogger or other spyware. Moreover, changing a password for a specific site is complicated because Password Multiplier requires users to remember additional information. Changing the master password also becomes tedious because the password on every site needs to be updated.

Built upon Password Multiplier and Petname Tool [7], Passpet turns a single master password into distinct passwords for different websites and uses petnames to help users recognize phishing attempts. In order to generate correct passwords, Passpet relies on a remote server to store site label files. However, Passpet has the same drawback as Password Multiplier in terms of master password vulnerability. Changing the master password is still tedious because a user needs to migrate passwords for every site. In addition, its remote storage server is vulnerable to various malicious attacks, which is acknowledged by the authors. We compare PasswordAgent with these existing systems and detail the advantages of PasswordAgent in Section 4.

3 Design of PasswordAgent

3.1 Overview

PasswordAgent consists of two major components: the Salt Repository and the Agent. The Repository stores salt lists enabling PasswordAgent to function transparently across either enterprise networks or the Internet. The Agent is used to retrieve the salts from the Repository, provide visual security indicators, and generate site specific passwords. In our design, each enterprise network maintains a Salt Repository providing salt storage services for its users. To achieve high reliability and scalability, it is possible that multiple servers function as the Salt Repository within one enterprise network. Usually, each user has a primary Salt Repository, but it is possible that one user has salt lists stored in multiple repositories.

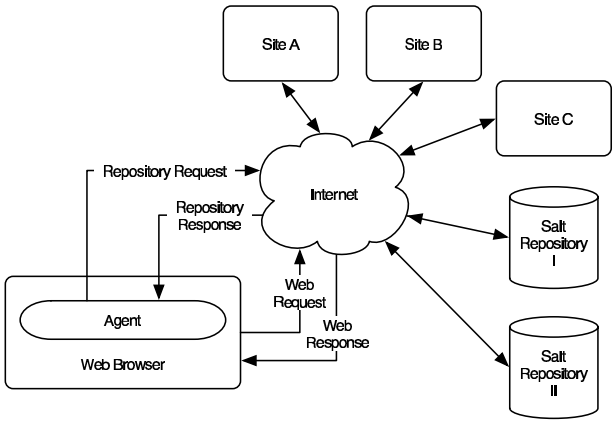


Figure 1: The architecture of PasswordAgent.

In contrast, the Agent is associated with each individual web browser as a browser extension. The basic architecture of the PasswordAgent is shown in Figure 1. Before continuing, it is important to have a grasp of the five different types of passwords discussed in this paper. Table 1 describes these passwords in detail.

Term	Description	Example
Plain-text Password	The user's password.	"secret"
Protected Password	The plain-text password but with additional data (either the activation hotkey or activation prefix) added to notify PasswordAgent that a site password needs to be generated.	"@@secret" OR [F2]"secret"
Site Password	The unique password generated for a site based on the site salt and plain-text password.	"2T7fYe10"
Agent Password	A password chosen by the user to protect their salts. It is only entered at the beginning of the browser session.	"likk@#0"
Repository Password	A password automatically generated from the Agent Password and used to authenticate to the Salt Repository.	"LT8@!dbn9"

Table 1: Password terminology.

To facilitate the deployment of the Salt Repository inside an enterprise network, the Repository can be integrated with any accessible web service that implements the Repository Interface. The web server can be publicly accessed via the Internet so that users can retrieve their salt lists from any location. The interface is a simple XML protocol that allows a user to register an account, save a domain and its associated salt, and retrieve a list of domains and salts. All the information in the salt list is encrypted before being stored. This not only guards against a compromised Repository, but also alleviates privacy concerns by making the domains inaccessible to anyone but the user. An overview of the data stored by the Salt Repository is illustrated in Figure 2.

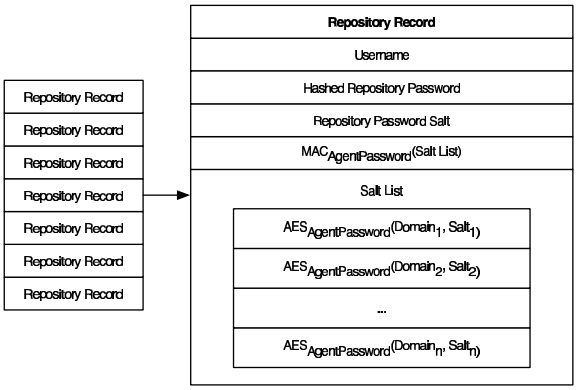


Figure 2: Data stored by the Salt Repository.

The Agent directly integrates into a web browser, allowing a user to generate passwords. The Agent is a variation of PwdHash for Firefox, but creating plug-ins for other major browsers should be a relatively simple task.

A PasswordAgent session begins like a normal browsing session with the user launching a web browser. The user must then log into the Agent with a username and Agent password. The Agent then transparently determines which Repository to use, and loads the user's salt list from it. The user continues to browse the web until a login form is encountered. Once the user enters a password field, the Agent indicates whether the current site is registered or not. If the user activates PasswordAgent, a unique site password will be generated by hashing the site's salt and the entered plain-text password. The login form is submitted with the site password, and the user logs into the site.

3.2 User Flow

Before using the PasswordAgent service, the user must register with a Repository and install the Agent. Registration consists of selecting a username and Agent password. Because the Agent is Repository agnostic, the username must provide enough information to determine which Repository to use. In consideration of this, all PasswordAgent usernames take the form of `username@domain` where `domain` is the domain name that the Repository belongs to. For example, a user who has the username `jsmith` and utilizes the institution of XYZ's PasswordAgent service would use the login `jsmith@xyz.edu`. The Agent can then locate the Repository at passwordagent.xyz.edu. This approach requires minimal memorization for a user, and allows for easy deployment and configuration of the Repository.

The Agent encrypts all the information that it stores in the Repository, so there is a requirement for both an encryption key and a password to authenticate to the Repository. To avoid having the user memorize two secrets, the Agent password is used as the encryption key and the Repository password is generated by hashing the full username (including the domain) and the Agent password together:

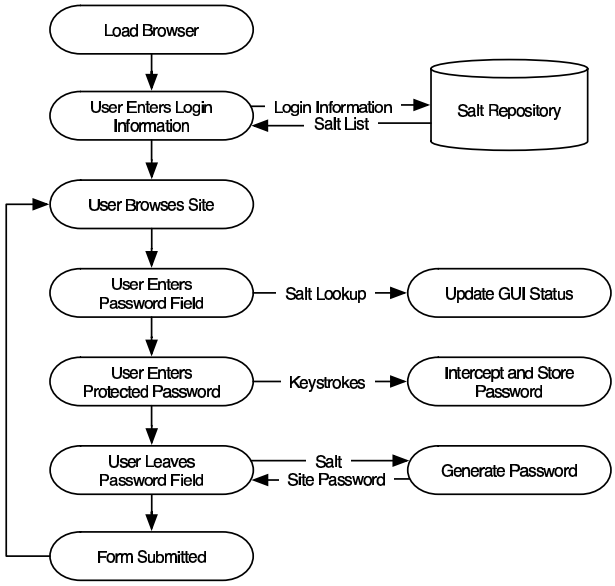


Figure 3: Basic user flow.

$$Password_{Repository} = SHA256(Password_{Agent}||Username) \quad (1)$$

$$EncryptedSalt = AES_{Password_{Agent}}(PlaintextSalt) \quad (2)$$

Since the Repository never knows the Agent password of a user, it cannot decrypt the stored information. In order to guarantee the integrity of the salt list, the Agent also stores a Message Authentication Code (MAC) [12, 13] calculated as:

$$MAC = HMAC - SHA256_{Password_{Agent}}(Domain_1||Salt_1 \\ ||Domain_2||Salt_2||...||Domain_n||Salt_n) \quad (3)$$

Registration can be accomplished either with automatic enrollment by a network administrator or via a web form provided by the Repository. Once registered for an account, a user would then need to install the browser plug-in. After installation, the user is ready to log in and begin a browsing session. An overview of a PasswordAgent session is given in Figure 3.

3.2.1 Login

When a web browser (e.g., Firefox) is first launched, the Agent lacks a salt list and is unable to protect any passwords. The user must authenticate to the Repository via

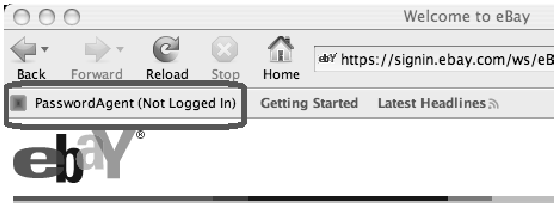


Figure 4: Toolbar displaying the status of PasswordAgent.

a login dialog. A toolbar button displayed by the Agent allows the user to enter the login dialog as shown in Figure 4.

Once the user enters a username and Agent password, the Agent determines the location of the Repository based on the domain portion of the username, generates the Repository password, and retrieves the salt list from the Repository. If successful, the salt list is decrypted using the user's Agent password and the MAC is verified. Should the MAC determine that the salt list has been tampered with, the user is warned via a dialog and the Agent remains logged out. Otherwise, the Agent updates the toolbar to reflect the new logged in status and retains the salt list in memory until the user logs out or exits the browser. To guard against fraudulent Repositories, all communication is performed over HTTPS. Because the Repository has to identify itself via an SSL certificate, the Agent is protected from being tricked into divulging the Repository password.

3.2.2 Browsing

If the user enters a password field during the process of web browsing, the Agent toolbar changes to inform the user whether or not the current site is registered. A site is considered registered if it has a salt associated with it, as shown in Figure 5(a), and unregistered if it does not, as shown in Figure 5(b). This allows the user to decide whether or not to enter a protected password.

3.2.3 Password Protection

The password input mechanism of the Agent is similar to that of PwdHash. In order to notify the Agent that a site password should be generated, the user enters a protected password. A protected password is created by either prefixing the plain-text password with @@ or by pressing a hotkey (F2). For example, a user who wishes to protect the plain-text password "secret" would type the protected password "@@secret" which would cause PasswordAgent to generate a site password. When a protected password is entered, PasswordAgent captures all keystrokes before they appear on the page, so JavaScript keyloggers cannot steal the plain-text password. Once the user leaves the password field, the protected password is analyzed to reveal the plain-text password (by removing the @@ prefix) and hashed together with the site's salt to create the site password. The site specific password is generated using the SHA256 hash function [3]:

$$Password_{Site} = SHA256(Password_{Plaint-text}||Salt_{Site}) \quad (4)$$

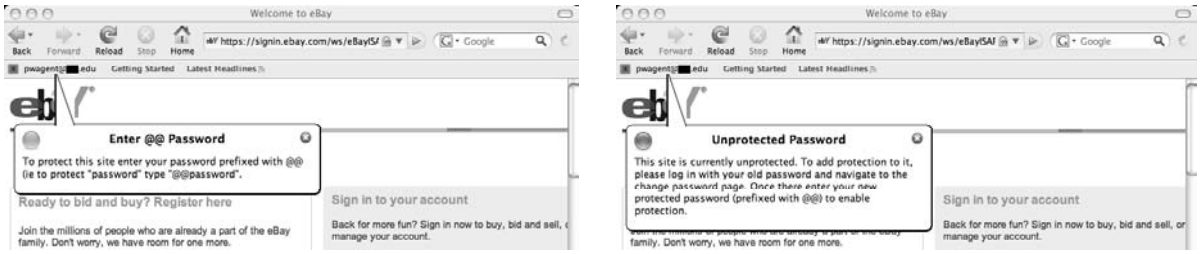
The main reason for using either the @@ prefix or the F2 hotkey is to let a user explicitly inform the hashing mechanism where to intercept the plain-text password. This guarantees that the data in other input fields will not be incorrectly hashed. The prefix @@ is chosen, because it is extremely unlikely that it will appear in a normal context. This enables PasswordAgent to scan the keystream and interpret @@ as an indicator to activate password protection. F2 is selected as a hotkey since it is currently not mapped to any functionality in Firefox [31].

3.2.4 Site Registration

To create a site password for a website, the site must first be registered in order to have a salt associated with it. If a user attempts to generate a site password via the prefix or hotkey mechanism on a site that has not yet been registered, an instructional dialog will appear. The dialog walks the user through registering the site with PasswordAgent. The dialog first confirms that the user wants to register the site with PasswordAgent, and hasn't accidentally triggered password protection. The user is then given a list of already registered sites as shown in Figure 6. The user is asked if the target site appears in this list. If it does, then the target site is actually a phishing attempt because it appears to be a registered site but in reality is from a different domain. The user is warned and prompted to navigate away. If the target site is not listed, the user is asked if he or she has an account with the target site or is creating a new account. Different instructions are displayed based on the user's response:

If the user has an existing account with the target site, that account must be migrated to use PasswordAgent. Migration is achieved by: (1) logging in with the plain-text password, (2) navigating to the change password page, and (3) entering the new protected password. A salt will then be generated, encrypted, and sent to the Salt Repository along with the updated MAC.

If the user is creating a new account on the unregistered site, then the user simply has to enter a protected password on the site registration form. The salt is generated, the new MAC for the salt list is calculated, and



(a) (b)

Figure 5: User focused password fields: (a) on a protected site, (b) on an unprotected site.

both are sent to the Repository for storage. The new salt is then used to generate the site password.

3.2.5 Multiple Accounts on One Site

A user can have multiple accounts on a single site, for example, someone may have two Gmail accounts. PasswordAgent is compatible with this scenario, as it can use the site's salt to hash both passwords. In this case, password uniqueness cannot be guaranteed because if the user selects the same password for both accounts, the protected password will also be the same. This is a minor issue, given that it is a relatively rare scenario. This issue also exists in PwdHash and Password Multiplier. Password protection is provided in that a compromised password on the site with multiple accounts will only effect that particular site - all other sites are guaranteed to have unique passwords.

3.2.6 Changing Site Password

The site password can be changed by one of two mechanisms. The first is to change the plain-text password as one would do with a normal password (i.e., "@@password" to "@@newpassword"). The Salt Repository does not need to be notified in this case, since the salt remains unchanged. The new password is protected in the same manner as the old password. This has the advantage of not requiring the user to learn any new paradigms about changing passwords. The second is to keep the plain-text password intact but to change the site salt.

3.2.7 Password Format

Every site has different requirements for passwords. Some sites require at least one non-alphanumeric character, while others prohibit them entirely. To allow for these different formats, the user's plain-text password is examined for clues as to the makeup of a valid site password. If the user does not include a non-alphanumeric character in the plain-text password, the site password would not contain one and the site would notify the

Figure 6: Information dialog that assists users in recognizing phishing sites.

user of the incorrect composition of the password. Any changes in the plain-text password will be reflected in the site password, enabling PasswordAgent to generate valid passwords for all sites without any specific prior knowledge. Such a design was first presented in [31]. While this technique does leak information about the plain-text password, it is of little concern because no information about the salt is revealed. This technique avoids the need to constantly update a list of composition rules for common sites on the Internet. This also addresses an important usability issue of users being dissatisfied with site passwords. Users become concerned when sites, like Hotmail, offer a password strength meter and the site passwords are rated as medium instead of strong [14]. By inspecting plain-text passwords for clues, the indicated strength of a password is directly related to the strength of the plain-text password. It should be noted that the actual strength of the site password is greater, even if the password meter indicates they are the same. A user provided character string has less entropy than a salted and hashed version of that string.

3.2.8 Roaming

Roaming can be achieved in one of two ways. A roaming user can either install the Agent as outlined before, or site passwords can be generated via a web interface

provided by the Repository. The web interface allows the user to log in and generate passwords for a specific site, which can be copied and pasted into the login form. This enables users without the ability to install the Agent to still access their accounts. PwdHash implements a similar web based mechanism for roaming users, however it is located at a URL that is complex and difficult to memorize [14]. Because the Repository is located at *passwordagent.domain*, it is simple to provide the web interface at that address. Doing this reduces the memory burden as users already know the domain of their Repository as part of their logins, all they have to do is remember to prepend *passwordagent* to it.

4 Security and Usability Analysis

4.1 Security Analysis

The primary goal of PasswordAgent, like other password hashing schemes, is to improve user security. Here we compare the security of PasswordAgent with those of LPWA, PwdHash, Password Multiplier, and Passpet in ten different aspects. The comparison results are summarized in Table 2. A detailed discussion is as follows, outlining the major security concerns with the existing password hashing mechanisms.

Unique Passwords: Each password hashing solution generates a unique password for each site, even if the plain-text password is the same.

Offline Attacks: PasswordAgent is less vulnerable to offline attacks. Because the salt list is not stored locally, launching an offline attack to retrieve the salt list is difficult. Moreover, the Salt Repository can defend against online attacks by limiting the number of login attempts allowed per minute. Password Multiplier and Passpet are also resistant to offline attacks as long as the local machine remains uncompromised. However, if an attacker breaches the computer and retrieves the cached master password, a relatively inexpensive offline attack can be launched to expose every site password. With PasswordAgent, even if the Agent password is stolen, only the salt list is revealed. The attacker would still need to launch an online attack against the target site to determine the site password.

Compromised Plain-text Password: In the scenario where the plain-text password is compromised, only PasswordAgent still provides user protection. An attacker would be unable to use the compromised password, because the random site salt is not known. PwdHash does not have this advantage, as the salt is the site's domain name, allowing an attacker to utilize the compromised password to access the site. Even worse, Password Multiplier and Passpet both use one plain-text password as a master password to generate all of the site passwords. Should the master password be compromised, every password protected by Password Multiplier and Passpet will also be compromised.

Compromised Site Password: All password hashing schemes claim to protect users when a site password is compromised. However, because PwdHash uses MD5 and a known salt, the domain name, it is possible to launch a brute force attack on the compromised password. A phisher impersonating a single site could launch a time-space trade-off attack and feasibly retrieve the plain-text password. In contrast, PasswordAgent defends against offline attacks with a large random salt. Assuming that it takes 1ms to calculate a hash with a 256-bit salt, it would take roughly 10^{66} years on average to find the plain-text password. Furthermore, even if attackers are able to recover the plain-text password, they still have to launch an online brute force attack in order to discover the salt for any other site that uses the same plain-text password.

Basic Phishing Protection: The nature of hash-based password generation allows all schemes to provide a basic level of phishing protection. Because each site password is unique, using any of these password generation tools on a phishing site will not immediately expose the login of the target site. As previously noted though, the site password can be used in offline attacks to reveal the plain-text password. LPWA, PwdHash, Password Multiplier, and Passpet all suffer from this problem. However, PasswordAgent offers the additional security with random salts, so even a stolen plain-text password will not give an attacker access to a login.

Advanced Phishing Protection: PasswordAgent provides early warning against phishing sites. If a user attempts to enter a protected password on an unregistered site, an information dialog notifies the user. This dialog, as shown in Figure 6, warns the user that the current site is not registered and displays a list of registered sites. This allows users to check if they are on a phishing site. Displaying security information in the browser chrome, PasswordAgent prevents its user interface from being spoofed by web pages. Because web pages do not have access to the browser chrome, it is difficult to place a fake login button or security indicator.

Shoulder Surfing Protection: PasswordAgent makes shoulder surfing—watching a user type in a password—much more difficult to succeed, because it requires the

		LPWA	PwdHash	Password Multiplier	Passpet	**PasswordAgent**
	Security					
1	Unique Password for Each Site	yes	yes	yes	yes	yes
2	Resist Offline Attacks	-	-	no	no	yes
3	Protect Compromised Plain-text Password	no	no	no	no	yes
4	Protect Compromised Site Password	yes	yes	yes	yes	yes
5	Basic Phishing Protection	yes	yes	yes	yes	yes
6	Advanced Phishing Protection	no	no	no	yes	yes
7	Enhance Shoulder Surfing Protection	no	no	no	no	yes
8	Secured Remote Storage	-	-	-	no	yes
9	Adaptation to Faster Computers	no	no	yes	yes	yes
10	Provide Data Anonymity	yes	no	no	no	no
	Usability					
1	Allow Easy Site Password Update	yes	yes	no	yes	yes
2	Notify if Site is Protected	no	no	no	yes	yes
3	Support all Site Specific Password Requirements	no	yes	no	no	yes
4	Minimal Change to Browsing Paradigm	yes	yes	no	no	yes
5	Requires 3rd Party Server	no	no	no	yes	yes

Table 2: Comparison of PasswordAgent with four other tools.

observation of two separate events, the typing of the Agent password and the typing of the site password. Since the Agent password is entered only when the user begins a session, an attacker is forced to hover around the victim for longer periods of time, increasing the chance of detection. Other schemes, however, only require one password, making the attacker easier to succeed.

Secured Remote Storage: The Salt Repository of PasswordAgent is cryptographically secure, and does not leak any useful information to attackers. By contrast, Passpet leaks not only whether a username exists (through the $list$ command) but also how large k_1 is [38], where k_1 is the number of iterations of a hash function used for generating the site password. The smaller the k_1, the weaker the password. Armed with this knowledge, an attacker can target a user with a small k_1 value and launch a brute force attack on the weakest master password. Both PasswordAgent and Passpet store only encrypted data and guarantee the integrity of the data with a MAC. Even in a situation in which a Salt Repository is compromised, the leaked information would not be useful because the attacker would have to brute force the salt list and then launch an online attack against the site specific passwords. It is technically possible to launch a brute force against the salt list, however it would take a prohibitively long time. This in combination with the required online attack against individual sites mitigates the possibility of a malicious Salt Repository compromising the security of PasswordAgent.

Adaptation to Faster Computers: PasswordAgent can adapt to faster computers and the associated greater power of attackers in launching dictionary/brute force attacks, by increasing the salt size. This is a minor change to the Agent implementation. The user simply regenerates a longer salt while keeping the plain-text password intact. The newly-generated site password is stronger, and no extra memory burden is placed on the user. In contrast, it is not easy for PwdHash to adapt to adversaries with more computing power. Both Passpet and Password Multiplier can increase the number of iterations to make it harder for an attacker to compute the site password.

Data Anonymity: Only LPWA has data anonymity as its goal. The other solutions, including PasswordAgent, focus solely on password protection. LPWA enables a user to browse, hold accounts, and email without ever revealing personal identification information.

4.2 Additional Usability Benefits

Usability is a key factor in any software system. A simple usability flaw might render a cryptographically secure system useless. Care is taken in the development of PasswordAgent to address usability concerns that exist in previous password hashing solutions. The specific usability benefits of PasswordAgent are detailed as follows.

Ease of Site Password Updating: PasswordAgent allows users to change their site passwords exactly like they normally do, via the change password page of the website. By changing it to a new protected password, users maintain all the benefits of PasswordAgent without any complicated or confusing processes. PwdHash has the same functionality. In contrast, Password Multiplier forces users to append information to the domain name being hashed. Not only is this confusing, but it also forces users to remember what additional information they are using for their logins [14]. Passpet uses a similar

mechanism, in which users can change the label of a site to change the password. Unlike Password Multiplier though, it remembers the change and does not require additional memorization.

Notification of Protected Sites: Only PasswordAgent and Passpet notify users when a site requires protected passwords. PasswordAgent displays a "notification bubble", which informs the user of the status of the site and how to login, as shown in Figures 5(a) and 5(b). In addition to notification bubble, PasswordAgent allows the user to view a list of all the registered sites. Both PwdHash and Password Multiplier fail to indicate whether a site is expecting a protected or plain-text password. Users who enter an incorrect password will often proceed to enter many of their other passwords, including plain-text passwords [14]. This leads to multiple passwords being exposed, a situation that is even worse than if no password protection is used.

Changing Master Password: The user can change the master password for the Salt Repository at any point without changing the password on any individual site. By entering the old and new Agent Password, the salt list can be decrypted and then re-encrypted with the new password. Because the same salts are used to generate the password, the site password remains the same. This is more convenient than in Passpet and Password Multiplier, where a change to the master password requires the user to login into each individual site and manually change the password.

Site Specific Password Requirements: Many sites have different password requirements, including size and acceptable characters. Only PasswordAgent and PwdHash examine the user's plain-text password for clues to the expected composition of a password. Any errors with the plain-text password are mirrored in the site password, so the user receives useful feedback.

Minimal Changes to Browsing Paradigm: Similar to PwdHash, PasswordAgent makes only minimal changes to the normal interaction between a user and a web browser. The only two changes include: (1) the user must log into the Agent when beginning a session, and (2) the protected password must start with @@ (or the user must activate PasswordAgent via the F2 key). These minimal changes should make the adoption of PasswordAgent easy. Password Multiplier and Passpet both require obvious deviations from the normal user login.

Ease of Switching Storage Servers: PasswordAgent is completely repository agnostic, and can easily transfer the salt list from one repository to another. In contrast, Passpet uses the storage server address as part of the master password generation, thus any change in the storage server address forces users to create a new master password and update all their site passwords.

```
<?xml version='1.0' encoding='utf-8'?>
<response>
  <status>statusInfo</status>
  <message>messageBody</message>
  ......
  <data>dataSection</data>
</response>
```

Figure 7: XML Response Format.

5 Implementation and Evaluation

5.1 Implementation

We build a prototype of PasswordAgent, in which the Salt Repository is implemented as a Java servlet and the Agent is implemented as a Firefox extension.

5.1.1 Salt Repository Interface

The Salt Repository Interface is a simple XML-based REST-style protocol, which allows the creation of user accounts, the updating of site salts, and the retrieval of the salt list. These methods fulfill the minimum requirements to maintain a salt list. The Interface is designed for ease of use with JavaScript's XMLHTTPRequest object. Because the XMLHTTPRequest object allows synchronous HTTPS requests and can translate an XML response into a DOM document, it takes minimal additional code for the Agent to communicate with the Repository. The Salt Repository is written as a Java servlet, which eases its deployment across different platforms. Any web server supporting HTTPS can serve as a Repository, as long as it implements the Salt Repository Interface and is located at *passwordagent.domain*.

Here the Salt Repository is maintained by a publicly accessible HTTPS server that implements the REST [20] methods as listed in Table 3. These methods allow users to maintain their salt lists.

The Interface is designed to be as simple as possible for implementation, and uses a simple XML response format that is easy to parse. The response format is illustrated in Figure 7. Each response contains at least a <status> element that is either "success" or "error" and a <message> element that includes a natural language description of the response. Some methods return

Method	Description	Parameters	Data Section Format
CreateUser	Creates a new user account.	user - the desired username. password - the desired repository password. hmac - the HMAC code to store for the current (empty) salt list	N/A
GetSites	Retrieves the salt list for a user.	user - the username of the user. password - the repository password of the user	<data hmac="SaltListHMAC"> <site domain="domain.com" salt="salt" /> <site domain="domain2.com" salt="salt2" /> </data>
SetSite	Stores a salt for a specified domain.	user - the username. password - the repository password. site - the domain salt - the new salt hmac - the HMAC code to store for the current (including this updated entry) salt list	N/A
UpdateUser	Updates a user's repository password.	user - the username. password - the repository password. newpassword - the desired new authentication password.	N/A

Table 3: Salt Repository methods.

a <data> section that includes more information, allowing further data to be passed to the caller.

5.1.2 Agent

The Agent is a Firefox extension written in JavaScript and XML User Interface Language (XUL) [10], without using native components. It is a modified version of the open source PwdHash. While the basic password protection activation code remains the same, additional functionality is provided in the form of a GUI, a more secure hash function, and a module to communicate with the Salt Repository. PwdHash has no visible GUI, PasswordAgent, by contrast, includes status indicators and warning dialogs to assist users in protecting their passwords. PwdHash uses the MD5 hash function, but recent collision attacks have rendered MD5 insecure [35]. PasswordAgent uses SHA256 for all hashing functions and AES [4, 15] for salt encryption. Although PasswordAgent uses a more complex hash algorithm and hashes larger values, it is still reasonably efficient as it takes only about 45ms to generate a password using SHA256, benchmarked on a 2.26Ghz Intel machine running SuSE Linux 10.2 with 512MB RAM.

5.2 Evaluation

We focus our evaluation on the usability of PasswordAgent, which is a key measure determining whether a password manager is really useful or even secure [14]. We choose user studies, i.e., laboratory user tests [32, 36], to assess the usability of PasswordAgent. We select PwdHash for a direct comparison with PasswordAgent. This is because both use the same activation method, and PwdHash scores higher than Password Multiplier on perceived security and usability [14]. In the design our usability tests, we follow a similar approach to the usability study on PwdHash and Password Multiplier [14].

Question	People Responding "Yes"
Do you sometimes reuse passwords on different sites?	96.4% (27)
Are you concerned about the security of passwords?	28.6% (8)
Criteria for choosing passwords:	
Easy to remember	75.0% (21)
Difficult for others to guess	42.9% (12)
Suggested by the system	0% (0)
Same as another password	57.1% (16)
Other	10.7% (3)
Participation in online activities requiring personal or financial details:	
Online purchases	75% (21)
Online banking	75% (21)
Online bill payments	28.6% (8)
Other activities	42.9% (12)
Do you use:	
A password manager?	3.6% (1)
A password generation tool?	0% (0)

Table 4: Participants' initial attitude towards password security.

5.2.1 Participants

There are 28 individuals ranging in age from 17 to 63 years old participated in the user study. Only one of the participants is a computer science major. None of the participants has any particular experience with computer security. A pre-task questionnaire, similar to the one in [14], is used to survey participants' initial attitude towards password security. The questions and responses are summarized in Table 4. We can see that only 42.9% of participants choose "difficult for others to guess" passwords, only 4% of participants do not reuse passwords across different websites, and only one participant has ever used software to manage passwords. A useful password generation tool would resolve the security issues caused by these poor password practices.

5.2.2 Tasks

Each participant is asked to complete a set of tasks using two password generation plugins: PasswordAgent

and PwdHash. The tasks are completed on two personal computers, designated A and B. Both computers run SUSE Linux and Mozilla Firefox. Computer A serves as the participants' primary computer, while Computer B is used to let participants install and use plugins from a computer other than their primary machine. Five tasks are carefully selected to reflect the realistic daily usage of a password generation plugin:

- **Migrate Login** : From computer A, logging on to a website W (Yahoo) with an account that has not yet been protected, migrating the account, and getting the password protected by the plugin. This task simulates taking an existing account and protecting it with the plugin.
- **Log Into Site** : From computer A, logging on to a website W (Google) with an account that has already been protected by the plugin. This task simulates a user's regular login process using protected accounts.
- **Update Password** : From computer A, logging on to the website W (Hotmail) with a protected account, and changing its password. This task simulates the process of changing the password of a protected account.
- **Second Login** : From computer A, logging on to a website W (Hotmail) with the protected account whose password has just been updated in "Update Password". This task simulates the process of logging in with updated passwords.
- **Remote Login** : From computer B, logging on to the website W (Amazon) with a protected account. This task simulates when users log in from remote machines that do not have the plugin installed.

Each task is performed with PwdHash and PasswordAgent. Participants are given a simple instruction sheet, which instructs them on how to use PwdHash and Password Agent. They are allowed to refer to the instructions whenever necessary. Accounts are created for the purpose of the usability tests, instead of having the participants use their personal accounts.

5.2.3 Results

Results are collected through both observation and questionnaires. An experimenter observes the test session of each participant and records the results. The experimenter does not provide additional instructions to a participant during the test session. The observed outcome of each task is classified into one of five groups: *successful*, i.e., the participant completes the task without a problem; *dangerous success*, i.e., the participant completes the task after an attempt that may lead to a security exposure; *failed*, i.e., the participant cannot complete the task and gives up; *false completion*, i.e., the participant erroneously thinks that the task has been correctly completed, when it has not; and *failed due to previous*, i.e., the participant does not complete this task due to the failure of previous task(s). Table 5 lists the task completion results for PasswordAgent and PwdHash. We can see that PasswordAgent achieves an over 90% success ratio for four tasks, and meanwhile it outperforms PwdHash in all the five tasks.

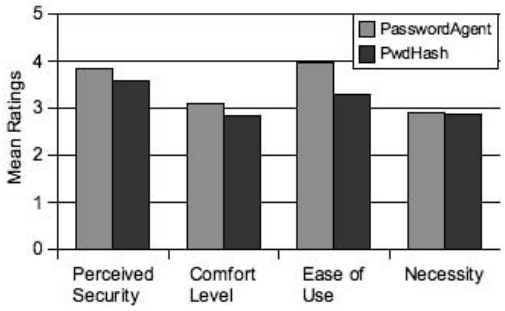


Figure 8: Mean questionnaire responses for each question group on scale of 1 to 5 (1 very negative, 3 neutral, 5 very positive).

After completing the tasks for a plugin, each user answers a questionnaire for that plugin. The questionnaire consists of eight Likert scale statements [28]. The participants are asked to indicate their degree of agreement with each statement after they finish the tasks. We use a five-point Likert scale: *strongly disagree*, *disagree*, *neutral*, *agree*, and *strongly agree*. Table 6 lists the questionnaire statements, which are very similar to the ones in [14]. A summary of the results are shown in Figure 8. The questionnaire focuses on four different categories: Perceived Security, Perceived Comfort, Perceived Ease of Use, and Perceived Necessity and Acceptance. While PasswordAgent scores higher than PwdHash in all four measurements, we further use t-test to determine the statistical significance of the differences in scores and observe that these differences do not have statistical significance.

6 Limitations

In this section, we discuss three limitations in PasswordAgent: vulnerability to keyloggers [21], the reliance on Salt Repository, and the usability limitations. PasswordAgent is designed to protect against web based attacks and cannot thwart compromises outside of the browser. Should a system have malicious software in-

Task	PasswordAgent					PwdHash				
	Success	Dangerous Success	Failures			Success	Dangerous Success	Failures		
			Failure	False Completion	Failed Previous			Failure	False Completion	Failed Previous
Migrate Login	92.9% (26)	0% (0)	7.1% (2)	0% (0)	0% (0)	75% (21)	14.3% (4)	10.7% (3)	0% (0)	0% (0)
Log Into Site	96.4% (27)	0% (0)	3.6% (1)	0% (0)	0% (0)	89.3% (25)	10.7% (3)	0% (0)	0% (0)	0% (0)
Update Password	96.4% (27)	0% (0)	3.6% (1)	0% (0)	0% (0)	67.8% (19)	14.3% (4)	17.9% (5)	0% (0)	0% (0)
Second Login	96.4% (27)	0% (0)	0% (0)	0% (0)	3.6% (1)	75% (21)	7.1% (2)	0% (0)	0% (0)	17.9% (5)
Remote Login	82.1% (23)	0% (0)	17.9% (5)	0% (0)	0% (0)	46.4% (13)	28.6% (8)	25% (7)	0% (0)	0% (0)

Table 5: Task completion results for PasswordAgent and PwdHash.

Perceived Security
My passwords are secure when using PasswordAgent.
I do not trust PasswordAgent to protect my passwords from cyber criminals.
Comfort Level with Giving Control of Passwords to a Program
I am uncomfortable with not knowing my actual passwords for a website.
Passwords are safer when users do not know their actual passwords.
Perceived Ease of Use
PasswordAgent is difficult to use.
I could easily log on to web sites and manage my passwords with PasswordAgent.
Perceived Necessity and Acceptance
I need to use PasswordAgent on my computer to protect my passwords.
My passwords are safe even without PasswordAgent.

Table 6: Post-task Questionnaire (for PasswordAgent, the questionnaire for PwdHash was identical other than the name of the software).

stalled such as spyware or a keylogger, both the Agent password and the individual site passwords can be compromised.

The Salt Repository is an important part of the PasswordAgent solution. Should it become unavailable (because of server issues, network problems, or DOS attacks), the user would be unable to log into any protected site. However, it is possible to use the Salt Repository as a backup, if the user's primary computer stores the salt list and then mirrors any changes to the Repository. This can achieve high reliability, but would come at a security cost. If the primary computer is compromised, the salt list has a higher chance of being exposed than before. A potential area for improvement would be the support of multiple synchronized repositories to prevent a single point of failure. Building such a mechanism is beyond the scope of this paper.

A user must activate the password protection by using @@ (the F2 key, or some other means). This is the main usability limitation that is common to PwdHash, Password Multiplier, and PasswordAgent. This extra activation step may make some users feel inconvenienced. Moreover, if a user forgets to invoke the protection, this limitation may lead to security exposures because the user's plain-text password might be sent to a phishing site [14]. Although the inconvenience still exists, the security risks caused by this limitation is eliminated in PasswordAgent. A phisher cannot obtain the correct site password since the salt is not accessible to the phisher.

Another usability limitation is that if a user forgets the Agent password, then there is no mechanism to retrieve the users salts. The user has to manually reset their passwords on each site, using a forgotten password feature. While inconvenient, most websites today provide a mechanism to reset forgotten passwords so serious harm is avoided.

7 Conclusion

We have developed PasswordAgent, an automatic password management system with enhanced hashing. PasswordAgent includes a Salt Repository and a browser plug-in Agent, and it provides a convenient and secure password protection service in an automatic manner. Without altering the normal interaction between a user and a login form, PasswordAgent automatically secures the user's plain-text password by rendering a unique site password for each website visited. Under the stronger security guarantee, a user's site password is robustly defended against password cracking and theft. We have implemented a prototype of PasswordAgent and conducted usability experiments. The evaluation results clearly indicate the usability benefits of PasswordAgent.

Acknowledgment

We would like to thank the anonymous reviewers and our shepherd Travis Campbell for their insightful comments. This work was partially supported by NSF grants CNS-0627339 and CNS-0627340.

References

[1] Anti-Phishing Working Group. http://www.antiphishing.org/index.html.

[2] Consumer sentinel network data book. Federal Trade Commission, February 2009.

[3] FIPS Publication 180-2. NIST, August 2002.

[4] FIPS Publication 197. NIST, November 2001.

[5] KeePass Password Safe. http://keepass.info/.

[6] Password strength. http://www.passwordmeter.com.

[7] Petname Tool. http://petname.mozdev.org/.

[8] RoboForm: Password Manager, Form Filler, Password Generator, Fill&Save Forms. http://www.roboform.com/.

[9] Windows Live ID. https://accountservices.passport.net/.

[10] XML User Interface Language (XUL) Project. http://www.mozilla.org/projects/xul/.

[11] ADAMS, A., AND SASSE, M. A. Users are not the enemy. *Commun. ACM 42*, 12 (1999), 40–46.

[12] BELLARE, M., CANETTI, R., AND KRAWCZYK, H. Keying hash functions for message authentication. In *Proceedings of Crypto'96* (1996), pp. 1–15.

[13] BELLARE, M., KILIAN, J., AND ROGAWAY, P. The security of the cipher block chaining message authentication code. In *Proceedings of Crypto'94* (1994), pp. 341–358.

[14] CHIASSON, S., VAN OORSCHOT, P., AND BIDDLE, R. A usability study and critique of two password managers. In *Proceedings of the 15th USENIX Security Symposium* (2006), pp. 1–16.

[15] DAEMEN, J., AND RIJMEN, V. The design of rijndael: Aes - the advanced encryption standard. *Springer-Verlag* (2002).

[16] DAVIS, D., MONROSE, F., AND REITER, M. K. On user choice in graphical password schemes. In *Proceedings of the 13th USENIX Security Symposium* (2004), pp. 151–164.

[17] DHAMIJA, R., AND PERRIG, A. Dejà vu: A user study using images for authentication. In *Proceedings of the 9th USENIX Security Symposium* (2000), pp. 45–58.

[18] DHAMIJA, R., TYGAR, J. D., AND HEARST, M. Why phishing works. In *Proceedings of the SIGCHI conference on Human Factors in computing systems* (2006), pp. 581–590.

[19] FELDMEIER, D. C., AND KARN, P. R. Unix password security - ten years later. In *Proceedings of Crypto'89* (1989), pp. 44–63.

[20] FIELDING, R. T., AND TAYLOR, R. N. Principled design of the modern web architecture. *ACM Transactions on Internet Technology (TOIT) 2*, 2 (2002), 115–150.

[21] FLORENCIO, D., AND HERLEY, C. Klassp: Entering passwords on a spyware infected machine using a shared-secret proxy. In *Proceedings of the 22nd Annual Computer Security Applications Conference (ACSAC'06)* (2006), pp. 67–76.

[22] FLORÊNCIO, D. A. F., AND HERLEY, C. A large-scale study of web password habits. In *Proceedings of the 16th International Conference on World Wide Web* (2007), pp. 657–666.

[23] GABBER, E., GIBBONS, P. B., KRISTOL, D. M., MATIAS, Y., AND MAYER, A. Consistent, yet anonymous, Web access with LPWA. *Commun. ACM 42*, 2 (1999), 42–47.

[24] HALDERMAN, J. A., WATERS, B., AND FELTEN, E. W. A convenient method for securely managing passwords. In *Proceedings of the 14th international conference on World Wide Web* (2005), pp. 471–479.

[25] HERLEY, C., VAN OORSCHOT, P., AND PATRICK, A. S. Passwords: If we're so smart, why are we still using them? In *Proceedings of the Financial Cryptography and Data Security Conference* (2009).

[26] JERMYN, I., MAYER, A., MONROSE, F., REITER, M. K., AND RUBIN, A. D. The design and analysis of graphical passwords. In *Proceedings of the 8th USENIX Security Symposium* (1999), pp. 1–14.

[27] KORMANN, D. P., AND RUBIN, A. D. Risks of the passport single signon protocol. *Comput. Networks 33*, 1-6 (2000), 51–58.

[28] LIKERT, R. A technique for the measurement of attitudes. *Archives of Psychology 140* (1932), 1–55.

[29] MORRIS, R., AND THOMPSON, K. Password security: a case history. *Commun. ACM 22*, 11 (1979), 594–597.

[30] RIVEST, R. L. The md5 message-digest algorithm. In *RFC 1320* (April 1992).

[31] ROSS, B., JACKSON, C., MIYAKE, N., BONEH, D., AND MITCHELL, J. C. Stronger password authentication using browser extensions. In *Proceedings of the 14th USENIX Security Symposium* (2005), pp. 17–32.

[32] RUBIN, J., AND CHISNELL, D. *Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests*. John Wiley & Sons, Inc., 1994.

[33] TARI, F., OZOK, A. A., AND HOLDEN, S. H. A comparison of perceived and real shoulder-surfing risks between alphanumeric and graphical passwords. In *Proceedings of the second symposium on Usable privacy and security (SOUPS '06)* (2006), pp. 56–66.

[34] THORPE, J., AND VAN OORSCHOT, P. Human-seeded attacks and exploiting hot-spots in graphical passwords. In *Proceedings of the 16th USENIX Security Symposium* (2007), pp. 103–118.

[35] WANG, X., AND YU, H. How to break md5 and other hash functions. In *Proceedings of EUROCRYPT 2005* (2005), pp. 19–35.

[36] WHITTEN, A., AND TYGAR, J. D. Why johnny can't encrypt: a usability evaluation of pgp 5.0. In *Proceedings of the 8th USENIX Security Symposium* (1999), pp. 169–184.

[37] YAN, J., BLACKWELL, A., ANDERSON, R., AND GRANT, A. Password memorability and security: Empirical results. *IEEE Security and Privacy 2*, 5 (2004), 25–31.

[38] YEE, K.-P., AND SITAKER, K. Passpet: convenient password management and phishing protection. In *Proceedings of the second symposium on Usable privacy and security (SOUPS'06)* (2006), pp. 32–43.

[39] YUE, C., AND WANG, H. Anti-phishing in offense and defense. In *Proceedings of the 24th Annual Computer Security Applications Conference (ACSAC'08)* (2008), pp. 345–354.

SEEdit: SELinux Security Policy Configuration System with Higher Level Language

Yuichi Nakamura
Hitachi Software Engineering Co., Ltd.
ynakam@hitachisoft.jp

Yoshiki Sameshima
Hitachi Software Engineering Co., Ltd.
same@hitachisoft.jp

Toshihiro Tabata
Okayama University
tabata@cs.okayama-u.ac.jp

Abstract

Security policy for SELinux is usually created by customizing a sample policy called refpolicy. However, describing and verifying security policy configurations is difficult because in refpolicy, there are more than 100,000 lines of configurations, thousands of elements such as permissions, macros and labels. The memory footprint of refpolicy which is around 5MB, is also a problem for resource constrained devices.

We propose a security policy configuration system SEEdit which facilitates creating security policy by a higher level language called SPDL and SPDL tools. SPDL reduces the number of permissions by integrated permissions and removes label configurations. SPDL tools generate security policy configurations from access logs and tool user's knowledge about applications. Experimental results on an embedded system and a PC system show that practical security policies are created by SEEdit, i.e., describing configurations is semiautomated, created security policies are composed of less than 500 lines of configurations, 100 configuration elements, and the memory footprint in the embedded system is less than 500KB.

Tags: security, security policy, configuration, SELinux

1 Introduction

Attackers can do everything in traditional Linux when they obtain the almighty root privilege by exploiting security holes in services running as root, or by exploiting vulnerabilities leading to privilege escalation[3][4]. To restrict such behavior of root, Security-Enhanced Linux (SELinux)[1][2] has mandatory access control feature; all processes including root processes can access resources only when a security policy permits the access. The mandatory access control model is called TE (Type-Enforcement)[5]. In TE, processes are assigned *domain* labels, and resources such as files and ports are assigned *type* labels, and what kind of domain can access what kind of type is described in a security policy. If the security policy is properly configured, all processes including root, attackers processes and viruses have only limited access rights. As a result, the damage by attackers and viruses is confined. Because of this confinement feature, SELinux is included in major Linux distributions[6], and is used for servers that require high level security. SELinux is also useful for network connected embedded devices such as cell phones and TVs. Actually, some Linux distributions for embedded system include SELinux[7].

To deploy SELinux to a system, a security policy must be created. The security policy is usually created by customizing a sample policy called refpolicy (Reference Policy)[8][9]. Refpolicy can be applied with almost no customization when configurations for applications in a target system are included in refpolicy. For example, refpolicy is almost perfectly configured for default applications included in Fedora and CentOS. However, customizing refpolicy is required for systems where refpolicy is not configured enough, such as embedded systems and systems where commercial applications are deployed.

There are three problems in the customization. First, it is difficult to describe configurations because there are more than 700 permissions and 1,000 macros. In addition, type labels must be associated with file names and network resources. Second, it is difficult to verify refpolicy. Since refpolicy is intended for multiple use cases, many configurations, more than 100,000 lines, are included. When engineers verify refpolicy before reuse, they have to review such a lot of configurations. Third is a problem of resource consumption. When SELinux is applied to resource constrained systems such as embedded systems, the files used and memory consumed by the security policy are a problem because refpolicy is large.

This paper proposes a security policy configuration system SELinux Policy Editor (SEEdit) that facilitates

creating security policy by a higher level language called Simplified Policy Description Language (SPDL) and SPDL tools.

- SPDL
  Instead of complicated macros, we propose a higher level language called SPDL. SPDL simplifies describing and verifying SELinux security policy configurations with two features. Firstly, integrated permissions in SPDL reduce the number of permissions by grouping related SELinux permissions. Secondly, it removes type configurations by identifying resources with names such as path name and port number.

- SPDL tools
  To solve the verification and size problems of refpolicy, the security policy is created by writing only the necessary configurations in SPDL without refpolicy. SPDL tools help the writing process by generating configurations using access logs and knowledge of users about applications.

The remaining of this paper is organized as follows. Problems in creating security policy (section 2), approaches of SEEdit to facilitate creating security policy (section 3) are explained. The detail of SEEdit (section 4), experimental results (section 5) are shown. Finally, related works (section 6), summary (section 7) and future works (section 8) are described.

2 Problems in creating security policy

In this section, problems in creating a security policy for a target system based on refpolicy are described after an overview of SELinux policy language and refpolicy.

2.1 SELinux policy language

The security policy is loaded to SELinux kernel in binary representation. However, it is hard to handle the binary security policy because it is unreadable for humans. To represent the security policy in text, SELinux has a basic policy language[10], which is mainly composed of the following four syntax elements.

(1) Assigning types
In SELinux, *type* labels must be assigned to resources to identify them. For example, the following statement is written to assign types to files.

<file name> system_u:object_r:<type>

Similar statements are used to assign types to network resources such as port numbers and NICs.

(2) Label declaration
Domains and types must be declared by *type* statements as shown below.

type <type or domain>, <attribute>;

<attribute> is used to inherit configurations which are described for <attribute>. For example, in the following statements, admin_t can read both httpcontent_t and ftpcontent_t.

```
type httpcontent_t, content;
type ftpcontent_t, content;
allow admin_t content:file read;
```

(3) Allowing access
The allow statement permits a domain to access a type as in the following syntax.

allow <domain> <type> <permission>;

<permission> is composed of *object classes* and *access vector permissions*. Object class means classification of resources such as file (normal file), dir (directory) and tcp_socket (TCP socket). For each object class, access vector permissions such as *read* and *write* are defined. For example, permission *file read* means reading normal files, *dir read* means reading directories.

(4) Conditional policy expression
To support multiple use cases in one security policy, SELinux policy language has conditional policy expressions as follows.

if(<parameter>){<statement>}

When <parameter> is true, then <statement> is enabled. For example, when CGI is necessary, the parameter *httpd_enable_cgi* is set true, and then accesses related to using CGI are permitted. Change of such parameters are applied without reloading security policy, because <statement> is embedded in the security policy.

2.2 Overview of refpolicy

To grant enough permissions for applications to work correctly, a lot of access rules should be described. In fact, the total number of access rules in a system often becomes more than 10,000, and sometimes exceeds 100,000. Therefore, it is not realistic to create security policy by writing configurations in SELinux policy language from nothing. To facilitate creating security policy, a sample policy called *refpolicy* is developed and

maintained by the SELinux community. Refpolicy is composed of macros and configurations for typical applications.

(1) Macros
M4[11] macros are defined to describe frequently used phrases in short words. Below is an example.

```
allow httpd_t contents_t r_file_perms;
define('r_file_perms','file { read
 getattr lock ioctl }')
```

r_file_perms is a macro, which is expanded to permissions related to reading regular files.

(2) Configurations for typical applications
Configurations for applications shipped with Linux distributions are prepared by the SELinux community and Linux distributors, and they are included in refpolicy. Figure 1 is part of the configuration for the http daemon. There are many macros, such as *init_daemon_domain*, *apache_content_template* and so on. In the figure, conditional expressions are omitted, but in fact, many conditional expressions are also included because refpolicy is intended to support as many use cases as possible, such as CGI, PHP and DB connection.

2.3 Problems in creating security policy using refpolicy

Customizing refpolicy is necessary when the use case of the system or its installed applications are beyond the expectations of refpolicy. For example, embedded systems and commercial applications are not within the scope of refpolicy. However, there are three problems in customizing refpolicy. One is the difficulty in describing configurations, second is the difficulty of verifying refpolicy and third is resource consumption.

2.3.1 Difficulty in describing configurations

The major difficulty in describing configurations is complicated configuration elements such as permissions, macros and types. The main reason of difficulty is the number of configuration elements. For example, there are more than 700 permissions and more than 1,000 macros and 1,000 types. In addition, nested macro definitions make understanding macros harder.

There are two more difficulties in types. First, engineers have to get used to types because in traditional Linux, they have been identifying files by file names not types. Secondly, there is also a problem of dependency in assigning new types. This problem is explained with an example. When the foo_t type is assigned under /foo directory and the bar_t domain is allowed to read the foo_t

```
# Assign httpd_t domain to http daemon
1  type httpd_t;
2  type httpd_exec_t;
3  role system_r types httpd_t;
4  init_daemon_domain(httpd_t,httpd_exec_t)
5 /usr/sbin/httpd -- gen_context(system_u
:object_r:httpd_exec_t,s0)
# Permit httpd_t to read /var/www
6  apache_content_template(sys)
7 /var/www(/.*)? gen_context(system_u
:object_r:httpd_sys_content_t,s0)
8  allow httpd_t httpd_sys_content_t:dir
 list_dir_perms;
9  read_files_pattern(httpd_t,httpd_sys_
content_t,httpd_sys_content_t)
10  read_lnk_files_pattern(httpd_t,httpd_
sys_content_t,httpd_sys_content_t)
# Permit httpd_t to wait connection on
  tcp port 80
11 corenet_all_recvfrom_unlabeled(httpd_t)
12 corenet_all_recvfrom_netlabel(httpd_t)
13 corenet_tcp_sendrecv_all_if(httpd_t)
14 corenet_udp_sendrecv_all_if(httpd_t)
15 corenet_tcp_sendrecv_all_nodes(httpd_t)
16 corenet_udp_sendrecv_all_nodes(httpd_t)
17 corenet_tcp_sendrecv_all_ports(httpd_t)
18 corenet_udp_sendrecv_all_ports(httpd_t)
19 corenet_tcp_bind_all_nodes(httpd_t)
20 corenet_tcp_bind_http_port(httpd_t)
21 gen_context(system_u:object_r:http_port
_t,s0)
```

Figure 1: Part of the configuration for the http daemon in refpolicy

type, the bar_t domain can read all files under the /foo directory. Next, if the foo2_t type is newly created, and assigned to the file /foo/foo2. the bar_t domain can not access /foo/foo2 because the bar_t domain is not allowed to access foo2_t. In this way, the bar_t domain was able to read /foo/foo2 before assigning the new type foo2_t, but bar_t can not access /foo/foo2 after the new type is assigned to /foo/foo2.

2.3.2 Difficulty in verifying refpolicy

For the purpose of Quality Assurance for a security policy which is created based on refpolicy, refpolicy should be verified. In this context, *verify* means understand what is configured, then find misconfigurations and modify them. However, it is difficult to verify because of the complexity of the configuration elements as stated before. In addition, the following points make verification more difficult.

- Amount of configurations

The size of refpolicy makes verification more difficult. For example, refpolicy included in Fedora 9 has configurations for almost all applications shipped with Fedora 9 and is composed of more than 2,000 types and more than 150,000 access rules.

- Conditional expressions
  Many conditional expressions are embedded in refpolicy, and they are sometimes included in macro definitions . Thus, it is difficult to figure out which configurations are enabled.

- Attributes
  Attributes are often used for types and they increase the time necessary to understand what configurations mean, as shown in the next example. The line *allow httpd_t httpdcontent:file read;* is included in refpolicy. *httpd_t* is a domain for the apache daemon, and *httpdcontent* is an attribute. To understand what kind of files httpd_t can access from the line, types that have the httpdcontent attribute have to be found by searching for type declaration statements, which are sometimes embedded in macro definitions.

2.3.3 Resource consumption

A security policy is saved as files in storage, then it is loaded to RAM at system boot. Therefore, the security policy consumes storage and RAM. Since refpolicy is intended for multiple use cases, many conditional expressions and configurations for many applications are included. As a result, the size of refpolicy becomes large. For example the refpolicy included in Fedora Core 6 consumes 1.4MB storage and 5.4MB RAM. In resource constrained systems such as embedded systems, this is a problem because they often have less than 64MB RAM and storage.

3 Approach to creating security policy

We propose a security policy configuration system SEEdit, which facilitates describing configurations, verifying a created security policy and creating a small security policy. The idea of the proposed system is explained in this section.

3.1 Higher level language: SPDL

The difficulty in describing configurations is caused by the large number of permissions, complicated macros and type configurations. Sophisticated macros can partly solve such problems, i.e., creating a small number of macros and removing nested macro definitions. However, type configurations are still necessary in such macros. Instead of macros, we propose a higher level language *SPDL* on top of SELinux policy language. SPDL aims to reduce the number of configuration elements by *integrated permissions* where related SELinux permissions are grouped. In addition, SPDL removes type configurations by identifying resources with their names. An example of configuration by SPDL is shown in Figure 2. The configured access rules are almost the same as Figure 1, but SPDL is simpler. Permissions related to reading files and directories are merged to integrated permission *r* and permissions to wait for connection on ports are merged to *server*. Additionally, names such as /var/www and port 80 are used to identify resources and assigning types to resources is not necessary. To apply SPDL configurations, the *SPDL converter* translates these configurations to SELinux policy language, i.e. SPDL converter generates the necessary type configurations, and expands integrated permissions to related SELinux permissions.

```
1 {
# Assign httpd_t domain to http daemon
2 domain httpd_t;
3 program /usr/sbin/httpd;
# Permit httpd_t to read /var/www
4 allow /var/www/** s,r;
# Permit httpd_t to wait connection on
  tcp port 80
5 allowcom -protocol tcp -port 80 server;
6 }
```

Figure 2: A configuration example of SPDL for http daemon.

The difficulty in verifying refpolicy is caused by two factors. First is the complicated configuration elements such as macros, permissions, attributes and conditional expressions. This complexity is solved by SPDL. Second is that many lines of configurations for access rules for applications not installed in the system and for rules disabled by conditional expressions are included. Our approach to solve the problem of many configuration lines is to describe only necessary configurations by SPDL without refpolicy, i.e. write configurations only for applications installed in the target system. Since neither conditional configurations nor configurations for unused applications are included, the number of configuration lines are expected to be reduced. This also contributes to reducing resource usage by the security policy.

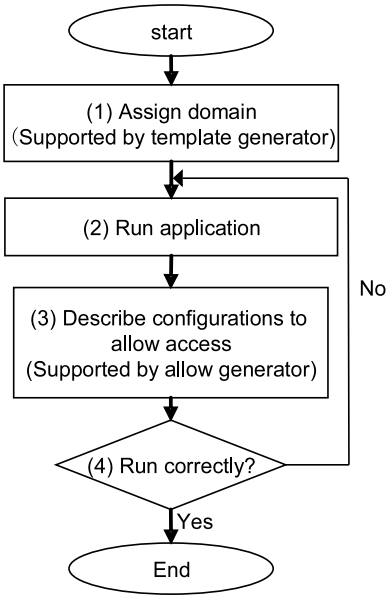


Figure 3: Typical process of creating a security policy

3.2 SPDL tools

In order to support writing configurations by SPDL without refpolicy, we propose SPDL tools composed of *template generator* and *allow generator*. SPDL tools aim to reduce the number of configurations written by hand during the process of creating a security policy.

Figure 3 shows a typical process of creating a security policy and this process is iterated for each target applications. (1) Configurations to assign a domain to a target application are described as in Figure 2 lines 2 and 3. (2) In order to figure out what kind of access rules should be described, access logs are obtained by running the target application. (3) Access rules are described using the access logs. For example, when an access log entry shows *foo_t domain read accessed filename bar* then an access rule that allows foo_t to read bar is described. (4) Run the application again and see whether it works correctly. If the application does not work correctly, run the application again and add configuration elements until the application works correctly.

Allow generator supports writing configurations allowing access in Figure 3 step (3). We adopt an approach of audit2allow[12] to automate describing configurations, i.e. generate configurations that allow accesses appearing in access logs.

Template generator outputs configurations in figure 3 step (1) by using configurations typical to application categories. For example, most daemon programs require access rights to create temporary files under /var/run and communicate with syslog. To produce more configurations, template generator uses the knowledge of the tool user about the target application, such as what kind of files and network resources the application accesses.

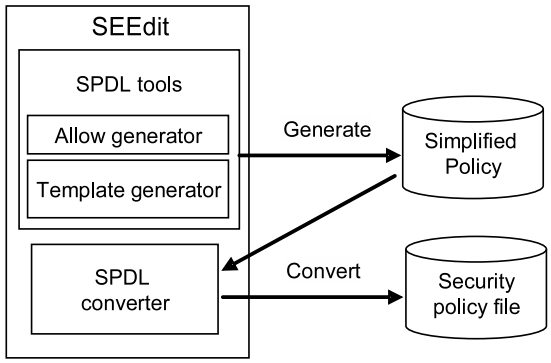


Figure 4: The architecture of SEEdit

4 Design and implementation of SEEdit

We designed and implemented SEEdit following the approaches discussed in the previous section. SEEdit is composed of SPDL tools and SPDL converter as shown in Figure 4. The security policy written in SPDL, called *simplified policy*, is created by a text editor or SPDL tools composed of allow generator and template generator. SPDL converter generates the security policy written in SELinux policy language from simplified policy. The design of SPDL and the implementation of SPDL converter and SPDL tools are described in the following subsections.

4.1 Design of SPDL

The main features of SPDL are integrated permissions to reduce the number of permissions, and configurations using resource names to remove type configurations. SPDL also has an *include* statement to reduce the number of lines. The detail is explained in this section.

4.1.1 Integrated permissions

While integrated permissions reduce the number of permissions by grouping permissions, permissions important for security should be kept. In order to include such important permissions, integrated permissions are designed from the viewpoint of protectiong the confidentiality, integrity and availability of a target system. Compromising confidentiality happens when an unexpected information goes out, and compromising integrity happens when an unexpected information comes into the

system. Thus, permissions related to input and output to files, network resources and IPCs have to be included in integrated permissions. The other permissions are privileges which can be abused to compromise availability and to facilitate attacks. For example, *setrlimit* permission that controls the resource usage limit of processes can lead to compromised availability. *cap_insmod* permission can result in installation of malicious kernel modules. Therefore, privileges have to be included in integrated permissions. The detail of integrated permissions are shown as follows.

(1) Integrated permissions for files
Integrated permissions for files are taken from previous research by Yamaguchi et.al[13] because they are designed to control input and output to files and directories. The integrated permissions are, *r* (read), *x* (execute), *s* (list directory), *o* (overwrite), *t* (change attribute), *a* (append), *c* (create), *e* (erase) and *w* (= o+t+a+c+e).

(2) Integrated permissions for network
Two integrated permissions related to input and output are designed for port numbers, NIC, IP address and RAW socket. For example, integrated permissions for port numbers are *server* (wait for a connection from outside) and *client* (begin a connection to outside).

(3) Integrated permissions for IPC
Integrated permissions for Sysv IPCs are *send* and *recv* to control input and output to processes. Integrated permissions for signals are designed to control sending each signal because SELinux can only control sending signals. For example, integrated permission *k* allows sending sigkill.

(4) Integrated permissions for other privileges
46 integrated permissions for other privileges are designed. Almost all permissions about privileges are included to prevent attackers from compromising availability and facilitating attacks. However, overlapped permissions are merged as an exception. For example, SELinux permission *capability net_admin* and *netlink_route_socket nlmsg_write* overlap each other because they are related to change kernel configuration of network. Thus, they are merged to the integrated permission *net_admin*.

4.1.2 Configurations using resource names

To remove type configurations, SPDL enables configurations using resource names. SPDL statements *allow* and *allownet* are designed as shown in Table 1 to enable name based configurations for files and network resources such as port number, NIC and IP address. To configure IPCs and other privileges, *allowcom* and *allowpriv* are also designed. Assigning types for IPCs and privileges is not required in SELinux, but they are shown for reference in Table 1.

```
domain httpd_t;
allow /var/www/** r;
```

Figure 5: Simplified Policy to be converted by SPDL converter

```
# Declare and assign type
1 type var_www_t;
2 /var/www(|/.*)
   system_u:object_r:var_www_t
#Allows permissions related to integrated
 permission r
3 allow httpd_t var_www_t:lnk_file { iotcl
  lock read  };
4 allow httpd_t var_www_t:file { iotcl
 lock read };
5 allow httpd_t var_www_t:fifo_file {
  iotcl lock read };
6 allow httpd_t var_www_t:sock_file {
  iotcl lock read };
```

Figure 6: Output of SPDL converter

4.1.3 Include statement

In order to reduce the number of configuration lines, the *include* statement imports configuration from a file.

#include *filename*;

For example, when the file daemon.te includes access rules commonly used for daemon applications, describing *#include daemon.te;* imports those access rules.

4.2 Implementation of SPDL converter

SPDL converter translates SPDL to SELinux policy language. The translation process is shown with an example of converting Simplified Policy in Figure 5 to configurations in Figure 6.

The httpd_t domain is allowed to read files and directories under /var/www in Figure 5. SPDL converter generates types from resource names. For example, it generates var_www_t type from filename /var/www, then outputs configuration to assign var_www_t under /var/www in the first two lines in Figure 6. Next, it generates configuration to allow access to the generated type as line 3-6 in Figure 6.

When different types are generated for files or directories under /var/www, accesses to such types are allowed. For example, when some domains are configured

Statement	Meaning	Example
allow *filename integrated permission*;	Allows access to *filename* using *integrated permission*.	*allow /foo/bar/** r;* permits to read files under /foo/bar directory.
allownet *resourcename integrated permission*;	Allows access to *resourcename* using *integrated permission*.	*allownet -protocol tcp -port 80 server;* permits to wait connection on tcp port 80.
allowcom *IPCname domain integratedpermission*;	Allows access to *domain* using IPC *IPCname* and communicate using *integrated permission*.	*allowcom -unix foo_t r;* permits to read data from process running as foo_t domain via unix domain socket.
allowpriv *integrated permission*;	Allows usage of privilege *integrated permission*	*allowpriv cap_sys_chroot;* permits to use chroot system call.

Table 1: Statements in SPDL to allow access to resources

*allow /var/www/cgi/** r;*, then configuration that assigns var_www_cgi_t to /var/www/cgi is generated. SPDL converter also generates configuration for httpd_t that allows reading var_www_cgi_t.

However, configurations using resource names do not work well for files dynamically created by processes. Dynamically created files mean files that are removed and created again. In SELinux, when a file is removed and created again, the type of the file is the same as the directory where it belongs. This behavior is sometimes a problem. For example, *allow /tmp/foo r;* is configured in foo_t domain. At first, /tmp/foo is assigned tmp_foo_t type, but when /tmp/foo is removed and created again, then the type is tmp_t. Therefore, the foo_t domain can no longer access /tmp/foo. To handle such cases, SPDL has *allowtmp* to configure assigning types correctly. The syntax of allowtmp is as follows.

allowtmp -dir *directory* -name *type integrated permission*;

This means files created under *directory* are assigned *type*. When *type* is *auto*, type is named automatically. For example, when foo_t domain creates temporary files under /tmp, we have to describe *allowtmp -dir /tmp -name auto r;* in foo_t domain, then type *foo_tmp_t* is generated and assigned to temporary files.

```
#Integrated permission
<macro value="allow_file_r"/>
#Corresponding SELinux permissions
  <secclass value="file" />
  <secclass value="lnk_file" />
  <secclass value="dir" />
  <permission value="read" />
...<snip>..
```

Figure 7: An example of permission mapping file

4.3 Implementation of SPDL tools

4.3.1 Allow generator

Allow generator outputs configurations that permit accesses recorded in the access log. The process is explained by an example below. First, allow generator reads SELinux access log, then extracts domain, resource name and permission from an access log entry. When a log entry is recorded that says *httpd_t domain process accessed filename /foo/bar whose type is foo_bar_t with permission file read*, httpd_t, /foo/bar/ and file read is extracted. The extracted information is not enough to create SPDL based configuration, because the permission is not an integrated permission. In order to obtain an integrated permission, allow generator converts SELinux permissions to integrated permissions by permission mapping, which contains mapping of integrated permission to SELinux permissions as illustrated in Figure 7. In the example, recorded SELinux permission is *file read*, then permission mappping is loooked up and corresponding integrated permission *allow_file_r* meaning integrated permission *r* for file is found. As a result, allow generator is able to output SPDL based configurations *allow /foo/bar/ r;*, from obtained domain, resource name and integrated permission.

4.3.2 Template generator

Template generator is implemented as a GUI. Figure 8 is a GUI to generate typical configurations. Users choose the profile of applications, and configurations are generated based on the profile. Figure 9 is a GUI to generate configurations from the user's knowledge. They can input their knowledge to the template generator without typing SPDL manually.

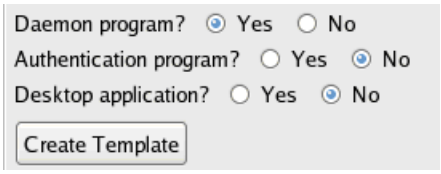


Figure 8: Template generator GUI to generate typical configurations

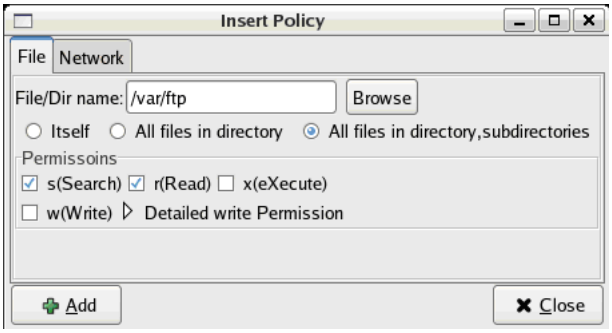


Figure 9: Template generator GUI to generate using knowledge of users

5 Evaluation

5.1 Experimental setup

In order to make sure whether SEEdit works, we used two typical systems for experiment. One is an embedded system configured for a small server, the other is a PC system configured for PC server as shown below.

(1) Embedded system

- CPU: SH7751R(SH4) 240MHz
- RAM: 64MB
- Storage: Flash ROM 64MB
- Linux distribution: not used
- SELinux: Linux 2.6.22
- Running services: httpd, vsftpd, syslogd, klogd, portmap

(2) PC system
Virtual machine (VMware 5.5) is used.

- Linux distribution: Cent OS 5 used for PC servers
- Running services: auditd, avahi_daemon, crond, cupsd, dhclient, gdm, httpd, klogd, mcstransd, named, ntpd, portmap, samba, sendmail, sshd, syslogd

Five domains are configured for services running on the embedded system, 16 domains are configured for services on the PC system. Access rules are written for these services to work properly. Memory usage of the security policy on the embedded system was also measured to evaluate whether SELinux is applicable to embedded systems. The memory consumption by SELinux was defined as the difference between memory usage when SELinux enabled and that when SELinux is disabled.

5.2 Result and consideration

In the experiment, we have successfully created security policies for both the embedded and the PC system. The process of describing configurations, verifying configurations and resource consumption are reviewed and considered. At last, trade-offs in SEEdit are also discussed.

5.2.1 Describing configurations

The first step to describe configuration is using template generator. To evaluate template generator, the assumption of knowledge on the part of the tool user is necessary because generated configurations depend on the user's knowledge. For evaluation, it is assumed that users know how to manage applications, i.e: they know file path of configuration files for applications, names of log files, names of content files which applications deliver and port numbers for applications. Assuming this, template generator produced 52% of the lines of configuration for the evaluation systems. For example, total 24 lines of configurations were described for http service in the PC system, and 12 lines were generated by template generator.

Next step is to produce configurations from access logs by allow generator. Most of the configurations generated by allow generator were able to be used without modification except for the following two cases. First is allow statements generated for dynamically created files. These allow statements have to be replaced with allowtmp statements. For example, foo_t domain dynamically creates and removes /tmp/foo, then log entry *foo_t domain write /tmp/foo* is recorded. Allow generator outputs *allow /tmp/foo w;* from the log entry. However, it should be replaced with *allowtmp -dir /tmp -name auto w;* as shown in section 4.2. Second is configurations generated from log entries which record access to normal files. Allow generator outputs *allow /var/www/index.html r;* for httpd_t from log entry *httpd_t read /var/www/index.html*. When the user knows http_t domain accesses /var/www directory, it is better to permit access to directory like *allow /var/www/** r;*. For the above two cases, the generated integrated permissions still can be used without modification.

	refpolicy	SPDL
File	130	9
Network	453	14
IPC	45	7
Privilege	80	46
Total	708	76

Table 2: Number of permissions in refpolicy and SPDL

As shown above, SPDL tools generate most parts of the configurations. In addition, to modify a generated SPDL configuration is easier than modifying refpolicy because the number of permissions are reduced as shown in Figure 2, complicated macros are not necessary, and type configurations are removed.

5.2.2 Verifying configurations

To verify created security policy, the difficulty depends on the number of configuration lines. The number of configuration lines in refpolicy is more than 100,000 with complicated permissions, macros and types, thus verification of refpolicy based security policy is difficult. On the other hand, in the experiment, the total lines of configuration are 174 for the embedded system, 401 for the PC system, and they are described with SPDL. Therefore, it is easier to verify configurations in SPDL than configurations in refpolicy.

Note that verifying configurations written in SPDL is meaningful as long as the output of SPDL converter is correct. Another work is necessary to ensure the result of SPDL converter. One possible way is a test tool. The tool inputs configurations in SPDL and is run for each domain defined in the configurations. Next the tool tries all access patterns to see if only accesses configured in the policy are permitted.

5.2.3 Resource consumption

The file size of the security policy in the embedded system is 71KB and RAM usage is 465 KB. In the system used in the experiment, storage is 64MB, RAM is 64MB. The consumption of storage and RAM is less than 1%. Thus, the created security policy is usable for the resource constrained embedded devices.

5.2.4 Trade-offs

There are two usability-security trade-offs in SEEdit. The first trade-off is integrated permissions used in SPDL because integrated permissions reduce granularity. For example, integrated permission for file *r* means read permissions for file, symlink and socket file. Therefore, allowing read access to symlink but not to file and directory can not be configured by *r* permission. This can be a problem in the embedded systems used in evaluation. In the embedded system, busybox[14] was used for system commands. In a system where busybox is installed, commands are executed via symbolic links to /bin/busybox(busybox executable). When /bin/ls is symbolic link to /bin/busybox and /bin/ls is executed, ls functions in /bin/busybox are called. If a domain foo_t needs access to busybox commands and is configured *allow /bin/** r;*, foo_t domain can access symbolic links under /bin, and foo_t can use busybox commands. However, if a confidential command file /bin/secret exists, foo_t can also access /bin/secret. If access to symbolic links were configured separately, foo_t would not be able to access /bin/secret. To solve this problem, the security policy generated by SPDL converter has to be edited. Another solution is to create a new statement in SPDL that enables configuring SELinux permissions directly.

The second trade-off is the audit2allow approach in allow generator. If there is a bug or malicious code in a program, and the program accesses files unnecessary for the program to work correctly, allow generator outputs configurations to permit access to such files. For example, if code that accesses confidential data is embedded in a CGI program by an evil programmer, then a configuration that permits access to the confidential data is outputted by allow generator after running the CGI. To prevent such a dangerous configuration to be included in the security policy, generated configurations should be checked by the SEEdit user. To help the check process, a tool that evaluates generated configurations would be useful.

6 Related work

Linux distribution Fedora includes security policy configuration tools called setroubleshoot [15], SLIDE [16] and system-config-selinux [17]. Setroubleshoot analyzes access logs and presents configurations when an application does not work due to SELinux access denial. SLIDE is an Integrated Development Environment (IDE) to configure refpolicy. It has features to aid describing configurations such as input completion. system-config-selinux is a tool to generate templates of configurations for new applications. It can generate templates using a wizard. The above tools are intended to aid configurations using refpolicy. The purpose is different from SEEdit because SEEdit does not use refpolicy.

polgen[18] is a security policy generator with a higher level language. Users of polgen first describe template configurations for the target applications using the language, then run the application. Next, polgen generates recommended security policy from access logs. The purpose of the higher level language of polgen is to de-

scribe template configurations, and users have to handle types and SELinux permissions after writing a template. The purpose is different from SEEdit because SPDL in SEEdit is intended to describe whole configurations.

SENG [19] is a higher level language for SELinux security policy. It is intended to replace m4 macros, not to reduce the number of configurations and remove type configurations.

Sellers et al.[20] also implemented a higher level language and IDE called CDS Framework[21]. It is also used in the FMAC[22] project in OpenSolaris. It enables configuration from the viewpoint of information flow control, but is not intended to simplify configurations.

There is also work related to the verification of security policy. Apol included in setools[23] has features to query security policy, such as querying what kind of types a domain can access. SLAT[24][25] is a system to analyze the security policy based on information flow goals. Analyzers describe an information goal, then SLAT finds violations of the information flow goal. Gokyo[26] analyzes the security policy based on Access Control Spaces, then suggests configurations which violate constraints. These tools are for SELinux policy language, but they can be applied to configurations which are converted from SPDL.

7 Summary

Security policy for SELinux is usually created by customizing a sample policy called refpolicy. However, creating security policy based on refpolicy has problems in describing and verifying configurations, and in resource consumption.

We have proposed a security policy configuration system SEEdit which makes creating security policy easier with a higher level language called SPDL and SPDL tools. SPDL reduces the number of permissions by integrated permissions, and removes type configurations by name based configurations. SPDL tools help in writing configuration by generating configurations based on access logs and the knowledge of tool users about applications. Experimental results on an embedded system and a PC system have shown that SEEdit resolves the problems creating security policy and practical security policy can be created with SEEdit.

8 Future work

There are remaining issues in ensuring the results of SPDL converter (section 5.2.2) and trade-offs in SEEdit (section 5.2.4). Another issue is co-existing with refpolicy. Currently SEEdit can not be used with refpolicy because type configurations generated by SPDL converter conflict with existing type configurations in refpolicy. SPDL converter has to be improved to resolve such conflicts.

9 Availability

SEEdit is available from sourceforge[27]. It is licensed under the GPL.

References

[1] Security-Enhanced Linux, http://www.nsa.gov/research/selinux/

[2] Loscocco, P. and Smalley, S.: Integrating Flexible Support for Security Policies into the Linux Operating System: Proc. FREENIX Track of the 2001 USENIX Annual Technical Conference, pp. 29 - 42 (2001)

[3] CVE-2008-0600: Common Vulnerabilities and Exposures (2008), http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-0600

[4] CVE-2007-5964: Common Vulnerabilities and Exposures (2007), http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2007-5964

[5] Boebert, W. E. and Kain, R. Y.: A Practical Alternative to Hierarchical Integrity Policies. Proc. the Eighth National Computer Security Conference, pp. 225-237 (1985)

[6] Coker, F., Coker,R.: Taking advantage of SELinux in Red Hat Enterprise Linux:Redhat magazine Issue 6 April 2005 (2005) , http://www.redhat.com/magazine/006apr05/features/selinux/

[7] Linuxdevices.com:MontaVista readies new Linux mobile phone OS (2007), http://www.linuxdevices.com/news/NS4364061392.html

[8] SELinux Reference Policy, http://oss.tresys.com/projects/refpolicy/

[9] PeBenito,C., Mayer,F., and MacMillan, K.:Reference Policy for Security Enhanced Linux.Proc. 2006 Security Enhanced Linux Symposium (2006), http://selinux-symposium.org/2006/papers/05-refpol.pdf

[10] Smalley,S. : Configuring the SELinux policy, NAI Labs Report #02-007, http://www.nsa.gov/research/selinux/docs.shtml

[11] GNU m4, http://www.gnu.org/software/m4/m4.html

[12] Linux man pages for audit2allow(1), http://linuxcommand.org/man_pages/audit2allow1.html

[13] Yamaguchi, T., Nakamura, Y. and Tabata, T: Integrated Access Permission: Secure and Simple Policy Description by Integration of File Access Vector Permission: Proc. The 2nd International Conference on Information Security and Assurance(ISA2008), pp. 40-45 (2008)

[14] Wells, N.: BusyBox: A Swiss Army Knife for Linux, Linux Journal, vol.2000, n.78es (2000)

[15] Denis, J.: Setroubleshoot: A User Friendly Tool to Diagnose AVC Denials: Proc. 2007 Security Enhanced Linux Symposium (2007), http://selinux-symposium.org/2007/papers/09-setroubleshoot.pdf

[16] SLIDE: http://oss.tresys.com/projects/slide

[17] Walsh, D.: A step-by-step guide to building a new SELinux policy module: Redhat magazine(2007), http://magazine.redhat.com/2007/08/21/

[18] Sniffen, B., Ramsdell, J. and Harris, D.: Guided Policy Generation for Application Authors:Proc 2006 Security Enhanced Linux Symposium (2006), http://selinux-symposium.org/2006/papers/14-guided-polgen.pdf

[19] Kuliniewicz, P.: SENG: An Enhanced Policy Language for SELinux: Proc 2006 Security Enhanced Linux Symposium (2006), http://selinux-symposium.org/2006/papers/09-SENG.pdf

[20] Sellers,C., Athey, J., Shimko, S. , Mayer, F. and MacMillan, K.: Experiences Implementing a Higher-Level Policy Language for SELinux: Proc 2006 Security Enhanced Linux Symposium (2006), http://selinux-symposium.org/2006/papers/08-higher-level-experience.pdf

[21] CDS Framework IDE, http://oss.tresys.com/projects/cdsframework

[22] OpenSolaris Project: Flexible Mandatory Access Control, http://www.opensolaris.org/os/project/fmac/

[23] SETools, http://oss.tresys.com/projects/setools

[24] Guttman, J., Herzog, A., Ramsdell, J. and Skorupka, C.: Verifying information goals in security-enhanced linux: Journal of Computer Security., 13(1), pp 115-134 (2005)

[25] MITRE Security-Enhanced Linux, http://www.mitre.org/tech/selinux/

[26] Jaeger, T., Edwards, A. and Zhang, X.: Managing access control policies using access control spaces: Proc the seventh ACM symposium on Access control models and technologies (SACMAT 02), pp. 3-12 (2002)

[27] SELinux Policy Editor Website, http://seedit.sourceforge.net/

An SSH-based toolkit for User-based Network Services

Joyita Sikder
Univ. of Illinois at Chicago

Manigandan Radhakrishnan
VMware

Jon A. Solworth
Univ. of Illinois at Chicago

Abstract

Network authentication, even when using libraries intended to simplify the task, is inordinately difficult. Separate libraries are used for cryptography, network authentication protocols, accessing stored authentication information, and verifying the identity of remote entities. In addition, service used must be authorized. Finally, privilege separation is needed to separate security sensitive, highly privileged operations from the remainder of the application.

These tasks consume thousands of lines of application source code (not counting the security libraries on which they rely), and require much specialized security knowledge from the application programmer and system administrator.

In this paper we present a simple toolkit called `sshUbns` which encapsulates all these tasks in an easy-to-use tool. We modified SSH to add in `sshUbns` (in addition to SSH's other modes) and implemented a new super-server called `unetd`. It reduces to a negligible level the amount of application server security code needed. This toolkit makes it easier to create secure networking code, reduces security specific knowledge needed by application programmers, and makes it easier for system administrators to protect and analyze their systems.

1 Introduction

Network service user authentication seems to be a simple procedure: The user provides either a password or some cryptographic proof of her identity to the remote service. The service verifies the user's identity, and authentication is complete.

In practice, however the task is far more complex:

- Passwords, if used, must be of sufficient diversity to prevent dictionary attacks. Since attackers today have access to large botnets, password attacks consisting of millions of guesses are easily possible, even if a host is blacklisted after a few tries. On the other hand, if cryptography is used it must be implemented correctly to prevent side channel attacks (thus exposing secret keys) and to ensure sufficient randomness of keys (preventing brute force attacks).
- Authentication must be mutual so that the user knows that she is talking to the legitimate service. This is typically done cryptographically, for example with RSA [20].
- To maintain authentication after the initial authentication protocol, cryptography is used to prevent undetected packet modification (and prevent viewing) in transit. Symmetric cryptography, such as AES [12], is used to provide these protections.
- If the service is not anonymous, it is necessary to authorize users. The user must be allowed to perform the service and the service's permissions must be tailored to those of the user.

The complexity is not limited to cryptographic algorithms and network protocols. In addition, a complex software stack is used. For example, Generic Security Services (GSS-API) transmits authentication tokens between client and server [17]; Network Services Switch (NSS) accesses the stored authentication information; and Pluggable Authentication Modules (PAM) actually authenticates the user [22]. Failures in the use and configuration of this software can violate authentication and authorization requirements. Ensuring that these tasks are properly done in traditional schemes requires examining each service's code and verifying that security services are properly used.

Finally, traditional mechanisms are implemented with libraries which share the address space of the application. When application logic and authentication sit in the

same address space, there is a danger that failures in application logic (e.g., buffer overflow) can cause authentication to fail—for example, by bypassing authentication all together. Moreover, these applications often need superuser privileges to bind to restricted ports or to change the user ID on whose behalf the service runs. Without careful partitioning, there is substantial code which runs with excess privileges. If this code is successfully attacked, these excess privileges increase the damage that the attacker can do.

To prevent these authentication failures, *privilege separation* is used [18, 7]. Privilege separation partitions logic over multiple processes so that most code runs with reduced privileges. Security sensitive code is isolated in a separate process with administrative privileges; the remaining parts of the application can then be run without administrative privileges. Using privilege separation, a highly privileged isolated process performs operations as a proxy for the application. This requires partitioning of the application and inter-process communication.

We consider here the most demanding of these problems, *User-Based Network Services (UBNS)* in which the service process operates with user-specific privileges, thus using the *Operating System (OS)* to restrict service accesses. UBNS services use OS access controls to limit the accesses that a service is allowed to do (by running user-specific parts of that service under the user's ID), and thus to isolate users from one another. Services such as mail, calendaring, distributed file systems, ftp, and source code revision control systems can be implemented as UBNS. Examples of UBNS services include `dovecot` for IMAP/POP3 mail delivery [1] and `zimbra` for calendaring [2]. Although such services can and have been built without UBNS, they require increased application-level authorization and pose greater dangers due to more application-level vulnerabilities [7].

UBNS is so demanding to implement, that often less secure mechanisms are used instead. For example, using traditional techniques `dovecot` requires 24,628 lines to support IMAP. Of that, over 9,307 lines of code are used for user authentication alone, some 37% of the total code base. In addition, to support privilege separation 4 different process types are used. Using new OS mechanisms, `netAuth` implemented UBNS functionality with only 5 lines (vs. 9,307 in the original `dovecot`) of application code [19]! In addition, the application code was simplified using a single process type (vs. 4 in the original), since privilege separation was provided by the implementation of the authentication. However, `netAuth` required OS kernel modifications and IPSec, and hence the code produced is not widely used.

Here, we describe a toolkit, `sshUbns` which provides almost the same functionality without OS kernel modifications. The `sshUbns` toolkit is built on top of *Secure Shell (SSH)* [26]. Unlike library-based approaches, `sshUbns` is implemented in two separate services, a modified SSH and `unetd`. It uses SSH's strong cryptographic authentication and cryptographic protection of communications over the network; it adds end-to-end security for networked applications. It provides strong protections needed for UBNS and yet is very simple to use. This simplicity is in three separate forms: (a) it is easier for system administrators to set and analyze protections; (b) there is less code for application programmers to write; and (c) higher level abstractions require less security expertise from the application programmer. Hence, the programmer and system administrator's task is simplified since the tool implements authentication, encryption and authorization.

Moreover, `sshUbns` is implemented using privilege separation. Like the kernel-based `netAuth`, `sshUbns` provides strong protections with a minimalistic programming interface. It allows system administrators to easily control who can use a service and to easily launch services, since these protections are provided in a service-independent way by the toolkit. Because it provides a simple toolkit for these important services, a system administrator's job of securing their system is vastly simplified. The `sshUbns` toolkit also supports the easier-to-implement class of services that are restricted to certain users but do not differentiate between authorized users, and hence may run as a pseudo user. However, we'll focus here on the support for UBNS.

The remainder of the paper is organized as follows. Section 2 describes related work. Section 3 describes SSH's port forwarding mechanism, which is the starting point for constructing `sshUbns`. Section 4 describes the `sshUbns` architecture. Section 5 measures the effectiveness of the implementation. Section 6 describes implementation alternatives and finally we conclude.

2 Related work

UBNS and privilege separation are two complementary ways to partition a service into multiple processes. Privilege separation is used to split an application into root and non-root processes. Both UBNS and privilege separation are design strategies to maximize the value of least privilege [21]. Retrofitting privilege separation is not difficult since root privileges are a super set of ordinary user privileges, and there exists both libraries [15] and compiler techniques [9] to do it. UBNS is more invasive as the privileges of different users overlap, and hence the protection of files and users which own processes must be carefully considered at the start of design.

SSH is a widely used UBNS service [26, 18], but is ill-suited to implement UBNS-based network services because of the way network services are built. In the net-

work case, the listening process exists before the connection is made and must know at connect time which user is associated with the service. SSH's port forwarding performs user authentication at the service host—but not at the service—and hence, to the service, the users of a host are undifferentiated. As a result, traditional UBNS services use authentication mechanisms such as SSL or passwords and OS mechanisms such as `setuid` which are awkward to program and may not be secure.

Alternatively, SSH allows a remote executable to be invoked, but that remote executable is not connected to a network service. Similarly, hg-login [3], as used in Mercurial, performs remote authentication using SSH, but `exec`s a new program rather than connect to a running network service.

In contrast, `sshUbns` both authenticates and authorizes the user, so that the service runs *only* with the permission of the user. Unlike SSH, `sshUbns` provides end-to-end security from client to service. The `stunnel` tool could have been used as an alternative to an SSH-based implementation—it provides similar protections to SSH port forwarding; the primary reason we chose SSH is because it uses a fixed port which is already allowed by our firewall rules.

The OKWS web server [16], built on top of the Asbestos OS [11] does a per-user demultiplex, so that each web server process is owned by a single user—it is another example of a UBNS. However, this facility is provided at the HTTP level via cookies, while the technique presented here is application (and application protocol) independent.

Kerberos [23] performs encryption using private key cryptography. Microsoft Windows' primary authentication mechanism is Kerberos. Kerberos works well in the enterprise, when the user it authenticates is part of the enterprise, but works less well in widely distributed systems. The problem in this setting is that the clients must be "kerberized versions". Kerberos does not directly support UBNS. Moreover, implementing Kerberos for an application is more complex, and less modular than `sshUbns`. Kerberos does have an advantage over our scheme in that it has a key distribution mechanism while SSH does not.

Distributed authentication consist of two components: a mechanism to authenticate the remote user and a means to change the ownership of a process. Traditionally, UNIX performs user authentication in a (user space) process and then sets the User ID by calling `setuid`. The process doing `setuid` needs to run as the superuser (administrative mode in Windows) [24]. To reduce the dangers of exploits using such highly privileged processes, Compartmented Mode Workstations divided root privileges into about 30 separate capabilities [6], including a SETUID capability. These capabilities were also adopted by the POSIX 1e draft standard [5], which was widely implemented, including in Linux.

Plan9's OS kernel uses a fine grained one-time-use capability [10], which allows a process owned by user U to change its owner to U'. It works with `factotum`, a user space process which actually performs the cryptography for the application. The `sshUbns` toolkit unlike Plan9 uses only generic POSIX mechanisms, and thus does not require kernel modifications.

Distributed Firewalls [14] (based on Keynote [8]) in contrast to SSH, implements per user authorization for services by adding it to the OS kernel implementation of `connect` and `accept` APIs. While Distributed Firewalls sit in front of the service, and thus are not integrated with the service, Virtual Private Services are integrated and thus can provide UBNS services [13], but unlike `sshUbns`, this relies upon kernel modifications.

3 SSH port forwarding

The closest service to `sshUbns` is SSH port forwarding. Using SSH, a command, executed by the user on the client

```
ssh -L 3000:localhost:25 example.com
```

results in the local port (3000) being tunneled to host `example.com` at port 25. The command is successful if `sshd` is running on `example.com`; the user has an account there; and port 25 is bound.

Now a process on the client can reach the service at port 25 at `example.com` by accessing port 3000 on the client. The connection between hosts is authenticated and cryptographically protected. The protection is coarse grained, since any user on the client may connect to port 3000—even those without accounts on `example.com`. Moreover, the service at port 25 (smtp) does not know which user is sending to it, although firewall rules can ensure that the port is only reachable from within `example.com`.

The above example assumes that the user name on the client is the same as on the server. If instead, the user's name at `example.com` is say, `dave`, then the SSH command would be:

```
ssh -L 3000:localhost:25 \
        dave@example.com
```

Although we shall assume the names match in the following text neither SSH nor `sshUbns` require this.

Figure 1 shows the traditional SSH port forwarding. (For simplicity, we leave out the server-based root-owned SSH processes which are used to establish the SSH connection). SSH authenticates and encrypts the traffic between client and server hosts. SSH ensures that the endpoints of the SSH tunnel are owned by the same user (U_2

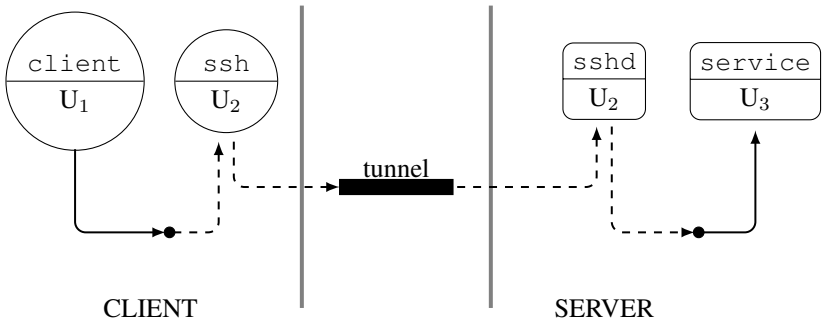


Figure 1: Client host to Service path using traditional SSH tunneling. Processes are indicated by circles or rounded edge rectangles. Above the interior line is the name of the executable, below the line is the user who owns the process.

in the Figure). However, because it is based on network ports—which don't perform any authentication—neither `client` to `ssh` network connection nor the `sshd` to `server` connection is authenticated. Thus traditional SSH is coarse grained, it is insufficient for UBNS such as mail, calendaring, etc. Moreover, since the ultimate user is unknown, logging effectiveness is very limited. Thus we turn to the architecture of `sshUbns`.

4 Architecture

In contrast to traditional SSH port forwarding, shown in Figure 1, `sshUbns` maintains the same user from client application to service, as shown in Figure 2. (Although the user is the same, the user name and user ID may be different on client and server, as per the previous section). It is this end-to-end property which ensures that the user is the same along the entire path which distinguishes `sshUbns` from SSH port forwarding.

The architecture we have implemented consists of three components:

SSH modifications which adds a UBNS mode to client and server sides,

unetd is a simple super-server which supports UBNS, and

server modifications which provide UBNS code to applications.

Of these, by design the server modifications are by far the smallest, since it minimizes the cost of porting servers to `sshUbns`. All the other code is independent of specific services.

4.1 SSH modifications

We have modified SSH to create a UBNS tunnel. This was done by modifying the port forwarding mode of SSH. The first step is to invoke SSH in UBNS mode from the client:

```
ssh -u -L 3000:localhost:25 \
          example.com
```

It is the "`-u`" which invokes `sshUbns`. (Alternatively, `autossh`—which automatically restarts SSH if there is a connection failure—can be used to make the connection robust even when the IP address changes).

We modified both the client side (`ssh`) and the server side (`sshd`) of SSH. On the client side, the `ssh` process which connects to the local port must be running *and* must be owned by the same user as the client process. This prevents other users on the client system (who don't have accounts on the server) from piggybacking on a legitimate user's port forwarding to the server system. Thus `sshUbns` is significantly safer than vanilla SSH port forwarding.

On the server side, we have written a `sshUbns` mode for `sshd` (based on its port forwarding mode) which interfaces with the service and runs on behalf of the remote user. It gets the port number of the user service process using a per service directory which is part of `unetd` (details are given in the next section).

For simplicity, we describe `sshd` as a process which runs on behalf of a user. Actually, to provide privilege separation `sshd` consists of two types of processes; one type which runs as root and the other as the user. However, only the user-owned process communicates with `unetd` and the UBNS.

TCP/IP are used everywhere except for a Unix domain socket between `sshUbns` components on the server side. Since the Unix domain socket is created in the file system, permissions can be (and in `sshUbns` are) set to ensure that the same user who creates the socket opens the existing socket. Unix domain sockets are not available on Windows computers, and in such a case it is possible to use TCP/IP sockets. However, where Unix domain sockets are available they are preferred.

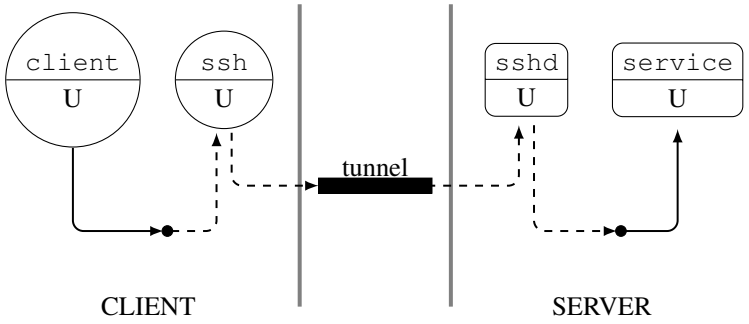


Figure 2: Client to host service path using `sshUbns`. The `client`, `ssh`, `sshd`, and `service` all run under the *same* user.

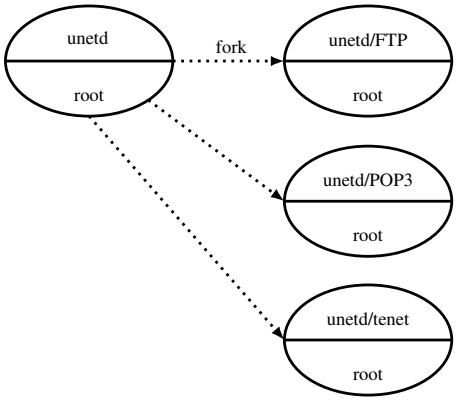


Figure 3: unetd and the service processes it spawns

For TCP/IP connections there is no standard method for user authentication and hence application-level protocols such as SSL are often used. However, when both ends of a TCP/IP connection are on the same host, it is possible to use OS calls to authenticate unmodified TCP/IP traffic. Although the method is non-standard across OSs, each of the major operating systems (Windows, Linux, Mac OSX) can determine the process and owner of the process which is at the other end of a local TCP/IP connection. For example, this information is available using `lsof` in UNIX-based systems or `openports` in Windows-based systems.

4.2 Unetd

We have written a daemon, `unetd` (for user-based network daemon) that launches UBNSs and authorizes users. Unetd is modeled after other super-servers such as `inetd` and `tcpd`. The configuration for `unetd` stored in `/etc/unetd/unetd.conf` contains lines of the form:

*port group * args*

The *port* (or service) specifies the desired service; the *group* specifies those users who are authorized for that service. The "*" is optional and means concurrent server, in which one process is spawned for each user connection. Without the "*" the server process for each user is sequential, meaning at any time there is at most one process per user. The *args* are the arguments with which `unetd/`*service* starts up (that is, `execs`) the *service*. Thus, our mechanism is sufficiently expressive to implement the primary different server types. We could also implement preforked servers, but believe `unetd` is sufficiently flexible without it.

Unetd runs as root, and creates a process per service. As a running example, we'll use POP3 as a service. For POP3, the created per-service `unetd` process is called `unetd/POP3` which listens to the port specified on its service configuration line. The service `unetd/POP3` does not contain any POP3-specific code, its purpose is to authenticate the user and direct the connection to the appropriate user-owned POP3 server.

It also checks that the user is authorized to use the service. When a POP3 `sshUbns` request arrives, `sshd` connects to the `unetd/POP3` and requests the port number of the POP3 process which is specific to that user. Finally, the POP3 process performs the user specific request, relying on the OS's access controls to ensure the accesses are appropriately authorized.

Figure 3 shows a `unetd` process which creates three different service processes, including `unetd/POP3`. All of the processes here are generic; the actual service (and the vast bulk of the code) is performed by user-based services that do not run with administrative permissions. Each service process is created to listen to a single inquiry port from `sshd` and launch the appropriate user based service.

Figure 4 shows the complete tree of processes created by `unetd`, including the service specific component. Each arrow indicates a process was forked. As can be clearly seen `unetd` is a UBNS and all server specific

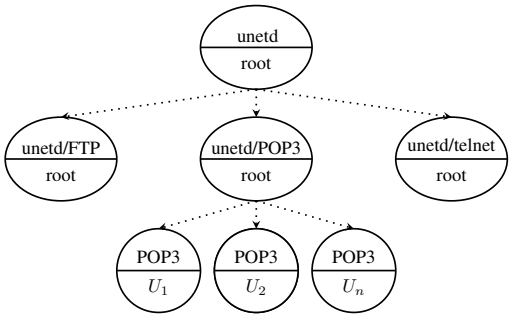


Figure 4: Three levels of processes created by `unetd`. For each process, a process identifier is shown on top, and the user ID on behalf of which the process runs (either root or ordinary users U_1, U_2, or U_n are shown).

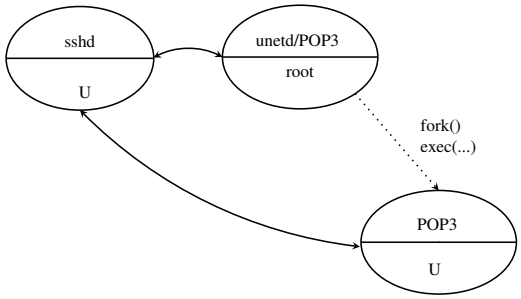


Figure 5: Creating the user-based service

code runs without root privileges.

4.3 Service support

It is trivial to modify a service for UBNS support. The port is opened by the parent process, so the only thing for the service to do is to check that the user of the user-based network service is the same as that of the process at the other end of the TCP/IP or Unix domain socket connection.

This checking is done by replacing the `accept` call with the `acceptUBNS` library call which does both accept and user ID checking. We note that this is the only security-specific call done by the service, the service has no need to deal with cryptography, authentication, user authorization, or privilege separation which are all generic services provided by sshUBNS.

The flow of service invocation on the server is shown in Figure 5. The `sshd` process sends to `unetd/POP3` its TCP port (having previously done a bind) and requests it to send it the port for user U's POP3 service. If none exists, or if POP3 is set up as a concurrent service, then a user-based service is created. Then `unetd/POP3` returns the port number. The `sshd` process then directly connects with POP3 server for U; U's POP3 server then does an `acceptUBNS` which ensures that `sshd` is owned by U thus completing the authentication.

4.4 End-to-end invocation of a user-based service

There is no change to `client` application code. The `client` configuration must specify the local port and local host to connect to `ssh` rather than directly to the service.

We consider the overall flow of a connection. Before this flow begins, we assume that (a) `ssh` in UBNS mode has been invoked on the client (and `sshd` has been started on the server) and (b) the server has invoked `unetd` which has started each UBNS, such as `unetd/POP3`. The overall flow from beginning to end of connection establishment is diagrammed in Figure 6. The trace of a connection is as follows:

1. the client application connects to `ssh` on the client,
2. `ssh` on the client connects to `sshd` on the server,
3. `sshd`
   (a) binds to a TCP/IP port p
   (b) sends p to `unetd/POP3` and asks for U's port address for POP3,
4. If the user is not in the group of users who are authorized to use that service, then `unetd/POP3` sends a failure message to `sshd`. Otherwise
5. If "*" has been specified in the configuration file or if there is no service for that user, then `unetd/POP3` does the following
   (a) a TCP/IP listening socket is created for the process to be forked,
   (b) a service process is `fork`ed and `exec`ed,
   (c) the UID of the resulting process is changed, and
   (d) the service executable is `exec`ed.
6. The `unetd/POP3` process replies back to `sshd` with the port number of the user's service process, and
7. `sshd` connects to the user's service process. which tests that it is coming from port p. Since `sshd` has been bound to port p, the connection must be from the specified user.

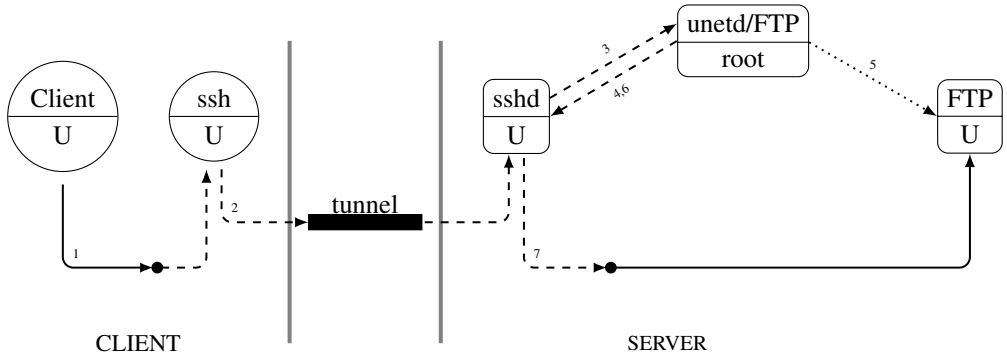


Figure 6: Overall flow

measure	time
real	0.004
user	0.000
sys	0.003

Figure 7: Time for a client response in seconds

measure	10 users
real time	3.390
user time	1.636
sys time	3.488
connections/second	295

Figure 8: Bandwidth measure (connections/second) and time for 10 users to each performed 100 POP3 connections

5 Experimental results

We have performed some initial testing of the performance of `sshUbns`. The testing was done on an AMID 4600+ 64x2 Dual Core in 64-bit mode. The software base is OpenSSH 5.1 (patch 1) and we ported `dovecot`'s POP3 server. Testing was done on a gigabit LAN. We used RSA keys.

In Figure 7 the client response time is shown for a remote use of POP3. The client side is much easier to measure than the server side, since we can simply measure the time for a trivial POP3 session, in which the only command is "bye". Total CPU elapse time is .004 seconds on the client side; we expect the server time to be slightly longer as it has an extra connection (to find the port of the UBNS). The first time `sshUbns` runs for a user it must also do a fork-exec of the user-owned service.

We have done some primitive tests to measure bandwidth using 10 users each doing 100 connections, the numbers are shown in Figure 8. The results show 295 connections per second on a dual core, or 147.5 connections per core per second. We have not had yet an opportunity to do any tuning which we expect will significantly increase performance.

We intend to do a number of ports to `sshUbns`, for example of web servers and calendaring systems.

6 Alternatives and future work

We would have liked to use UNIX Sockets throughout. This would have removed the need to do an `acceptUBNS` instead of an `accept` and authorization to connect to the UNIX socket could be done by the UNIX socket mechanism. Unix domain sockets are also considerably faster than using `lsof`. This performance advantage is far more important on the server side, and hence we have assumed it for our server experiments.

Using UNIX Sockets is considerably less attractive on the client side, since (1) client software many not be under the control of the server organization, (2) client software may be proprietary and hence not easily modified, and (3) there may be many different implementations of client software (e.g., many different mail user agents) thus increasing the difficulty of modifying them. The performance issues for the client are small, since each client is expected to use only a relatively small number of `sshUbns` connections.

SSH can be set up to be based on public key only, or to allow a combination of public key and password. Public key authentication is more secure, but requires some method for installing the public keys on the servers.

We could have used the ability to transfer a file descriptor over a UNIX socket to make `unetd/POP3` send the connection transparently to POP3 service for that user. This would allow local (i.e., non-networked) clients

to connect transparently to a UBNS. We will implement this in the next version of our software.

The current implementation requires patching `sshd` and possibly to the application (if we choose to use UNIX sockets for communication). A less invasive approach would be to make these changes as part of some library or wrappers (like TCP Wrapper [25]) that are linked with the program. This imposes difficulties for two reasons: (1) the communication between `sshd` and `unetd/POP3` and between POP3 and `unetd/POP3` has to be done in the library or wrapper and as part of `accept` or `connect` and (2) both `sshd` and the application (POP3) may be required to perform security critical operations before and/or the establishment of a connection that may not be securely performed without patching the application.

It would be interesting to extend this mechanism to applications which don't easily support port redirection (e.g., some web servers). Since the ports are not known in advance, some mechanism would be needed to examine packets without redirection; we are considering using TUN/TAP for this interface [4]. The TUN interface would also make invoking `sshUbns` transparent on the client.

The design of `sshUbns` is intended to be able to run on Windows as well as Unix-based hosts. We have used Unix domain sockets in only one single place, on the server side. To port this code to a Window's server, it would be necessary to use some other form of IPC, for example TCP/IP and to use `openports` for authentication between `sshd` and `unetd/POP3`. Similarly, `openports` could be used on the client side for connection between the application and `ssh`.

We have not made any attempt to make `sshUbns` fast. For large configurations, the cost of doing these operations may be significant, and performance optimization important. This is left for future work.

7 Conclusion

Often, it is assumed that security must be traded off against other properties such as usability or code complexity. Sometimes, however, we pay a far higher price for security than is necessary, largely because of the history of incrementally adding security. Comprehensive toolkits—which manage a set of related security issues—can have significantly lower overall complexity than a piecemeal approach while attaining strong security.

We built this tool because we wanted a better way of using and constructing authenticated services. The toolkit, `sshUbns`, is painless to use as it requires only a single line of code in an application to provide authentication, authorization, encryption. It is privilege separated, thus isolating security sensitive operations from the application. Issues of key size, authentication method, and many other issues become irrelevant for the application programmer. However, porting code is more involved because of the large number of lines of code which must be removed from legacy code. We plan to do several more ports.

The `sshUbns` toolkit is particularly attractive for system administrators. First, system administrators are adept at configuring solutions from tools. Second, `sshUbns` is general purpose and thus applicable to a whole range of networked applications. Third, it builds on well known tools and concepts, notably SSH and super-servers. Fourth, it avoids much of the need to individually examine application code and configurations to determine setting, a time consuming and unfortunately error prone process. Fifth, it is consistent across applications, reducing user education and system documentation issues.

Acknowledgements The program committee reviewers provided detailed, extensive, and useful comments. Two of the reviewers served as shepherds for the paper. David Plonka did an amazing and energetic job of providing many notes, giving us suggestions, and keeping us on schedule. William LeFebvre provided useful comments and kept us centered on the most important issues. Thanks to Wenyuan Fei, Prasad Patil, and Michelle Zhou for proofreading.

References

[1] http://www.dovecot.org/.

[2] http://www.zimbra.com/.

[3] http://www.selenic.com/mercurial/wiki/index.cgi/SharedSSH.

[4] vtun.sourceforge.net/tun/.

[5] IEEE/ANSI Draft Std. 1003.1e. Draft Standard for Information Technology–POSIX Part 1: System API: Protection, Audit and Control Interface, 1997.

[6] Jeffrey L. Berger, Jeffrey Picciotto, John P. L. Woodward, and Paul T. Cummings. Compartmented mode workstation: Prototype highlights. *IEEE Transactions on Software Engineering*, 16(6):608–618, 1990. Special Section on Security and Privacy.

[7] Daniel J. Bernstein. Some thoughts on security after ten years of qmail 1.0. In *First Computer Security Architecture Workshop*, page 1. ACM, 2007. Invited paper.

[8] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. RFC 2704: The KeyNote Trust-Management System Version 2, September 1999.

[9] David Brumley and Dawn Xiaodong Song. Privtrans: Automatically partitioning programs for privilege separation. In *USENIX Security Symposium*, pages 57–72, 2004.

[10] Russ Cox, Eric Grosse, Rob Pike, Dave Presotto, and Sean Quinlan. Security in Plan 9. In *Proc. of the USENIX Security Symposium*, pages 3–16, 2002.

[11] Petros Efstathopoulos, Maxwell Krohn, Steve VanDeBogart, Cliff Frey, David Ziegler, Eddie Kohler, David Mazières, Frans Kaashoek, and Robert Morris. Labels and event processes in the asbestos operating system. *SIGOPS Oper. Syst. Rev.*, 39(5):17–30, 2005.

[12] FIPS. *Advanced Encryption Standard (AES)*. National Institute for Standards and Technology, pub-NIST:adr, November 2001.

[13] Sotiris Ioannidis, Steven M. Bellovin, John Ioannidis, Angelos D. Keromytis, and Jonathan M. Smith. Virtual private services: Coordinated policy enforcement for distributed applications. *IJNS*, 4(1), January 2007. `http://www1.cs.columbia.edu/~angelos/Papers/2006/ijns.pdf`.

[14] Sotiris Ioannidis, Angelos D. Keromytis, Steve M. Bellovin, and Jonathan M. Smith. Implementing a distributed firewall. In *Proceedings of the 7th ACM conference on Computer and Communications Security*, pages 190–199. ACM Press, 2000.

[15] Douglas Kilpatrick. Privman: A library for partitioning applications. In *USENIX Annual Technical Conference, FREENIX Track*, pages 273–284. USENIX, 2003.

[16] Maxwell N. Krohn. Building secure high-performance web services with OKWS. In *USENIX Annual Technical Conference, General Track*, pages 185–198, 2004.

[17] John Linn. Generic interface to security services. *Computer Communications*, 17(7):476–482, July 1994.

[18] Niels Provos, Markus Friedl, and Peter Honeyman. Preventing privilege escalation. In *Proceedings of the 12th USENIX Security Symposium*, pages 231–242. USENIX, August 2003.

[19] Manigandan Radhakrishnan and Jon A. Solworth. NetAuth: Supporting user-based network services. In *Usenix Security*, pages 227–242, 2008.

[20] Ronald Rivest, Adi Shamir, and L. Adleman. On digital signatures and public key cryptosystems. *Communications of the ACM (CACM)*, 21:120–126, 1978.

[21] J. H. Saltzer and M. D. Schroeder. The protection of information in computer system. *Proceedings of the IEEE*, 63(9):1278–1308, 1975.

[22] Vipin Samar. Unified login with Pluggable Authentication Modules (PAM). In Clifford Neuman, editor, *Proc. ACM Conference on Computer and Communications Security (CCS)*, pages 1–10. ACM Press, 1996.

[23] Jennifer G. Steiner, B. Clifford Neuman, and J. I. Schiller. Kerberos: An authentication service for open network systems. In *Winter 1988 USENIX Conference*, pages 191–201, Dallas, TX, 1988.

[24] W. Richard Stevens. *Advanced Programming in the UNIX Environment*. Addison-Wesley, 1992.

[25] Wietse Venema. TCP WRAPPER: Network monitoring, access control and booby traps. In *Proceedings of the UNIX Security III Symposium*, pages 85–92, Baltimore, MY, USA, September 1992. USENIX Association.

[26] Tatu Ylonen. SSH—secure login connections over the Internet. In *Proc. of the USENIX Security Symposium*, pages 37–42, San Jose, California, 1996.

Federated Access Control and Workflow Enforcement in Systems Configuration

Bart Vanbrabant, Thomas Delaet and Wouter Joosen
{*bart.vanbrabant, thomas.delaet, wouter.joosen*}*@cs.kuleuven.be*
DistriNet, Dept. of Computer Science,
K.U.Leuven, Belgium

Abstract

Every organization with more than a few system administrators has policies in place. These policies define who is allowed to change what aspects of the configuration of a computer infrastructure. Although many system configuration tools are available for automating configuration changes in an infrastructure, very little work has been done to enforce the policies dealing with access control and workflow of configuration changes. In this paper, we present ACHEL. ACHEL makes it possible to integrate fine-grained access control into existing configuration tools and to enforce an organization's configuration changes workflow. In addition, we prototype ACHEL on a popular configuration tool and demonstrate its capabilities in two case studies.

1 Introduction

Because the scale of modern computer infrastructure keeps increasing, so automation has become a crucial part of system configuration. Tools are important in system configuration for increasing the automation and autonomy of computer infrastructures. These tools have contributed to successfully scaling infrastructures without a linear growth in manual system administration [11].

A typical system configuration tool [9] translates a configuration specification to a per-system profile. Such a profile describes the desired state of a managed system. A local component of the system configuration tool checks whether the current state of the target system matches the intended profile and makes adjustments if necessary [10, 14, 19, 25, 32]. This local component is called the *deployment engine* of a system configuration tool. The configuration specification for the configuration tool is often retrieved from a central version-controlled repository [9, 21, 22, 33] such as CVS or Subversion. Such a repository provides a full history of the infrastructure's configuration. All configuration changes are directly checked into this central configuration specification repository.

Managing an infrastructure based on a centrally available configuration specification comes at a price. Because the central specification controls all aspects of all managed systems, control over the specification means

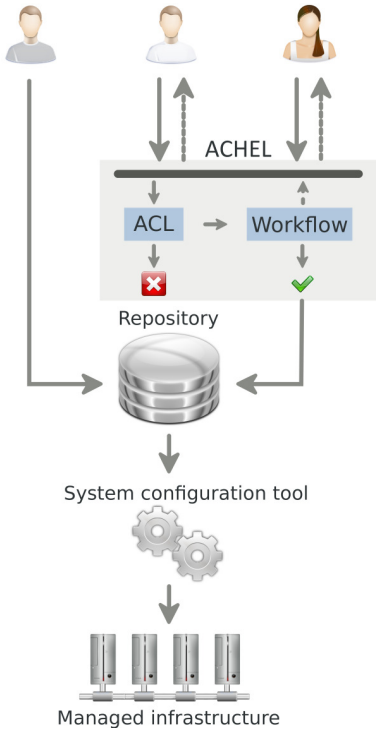


Figure 1: Conceptual overview of systems configuration without and with ACHEL.

control over all managed systems in the infrastructure. From a security perspective, the configuration specification repository requires very strict access control. Unfortunately, most existing tools do not provide any access control mechanisms, but rather reuse the access control available in the revision control repository [10, 14, 17, 19, 25, 32]. The access control systems of these revision control repositories default to allowing or denying full access based on the credentials of the user.

Although the hardware and the software in all infrastructures is quite comparable, the organization of system administration varies between different infrastructures [9, 22]. These differences can be attributed to system administration being organized either in a central or in a federated manner, as well as to the existence of different policies: certain infrastructures integrate changes directly into the configuration specification, others require changes to be approved by management, and yet others require rigorous quality control before a change is deployed. It is very hard to support these *workflows* in existing systems because of the limited workflow enforcement supportthat these systems provide [6, 16, 28].

In this paper we present ACHEL, a framework that enables the integration of fine-grained access control into existing configuration tools and enforces configuration change workflows in federated infrastructures. Figure 1 shows a conceptual overview of the process of updating a configuration specification, both with and without ACHEL. To improve the expressiveness of the access control rules, these rules are defined at the same abstraction level as the configuration specification language supported by the system configuration tool. ACHEL achieves this by taking the structure and the meaning of the configuration specification into account when generating *semantically meaningful changes*, instead of identifying changes line by line, such as the common diff-algorithm used in source code management does. ACHEL uses these semantically meaningful changes and the author history of each configuration statement to define fine-grained per user access control rules. Because a large portion of ACHEL is language agnostic, support for new configuration specification languages can be added with limited effort, as we did in our prototype.

ACHEL's second contribution is that it enforces workflow on configuration specification changes between repositories. We define a workflow as a set of rules that defines what steps a change should go through before it can be deployed on the managed infrastructure. ACHEL provides flexible workflows for centralized and federated infrastructures by building on a distributed version control system [2] and combining it with our fine-grained access control rules. A service or user signals its approval of a change by digitally signing the globally unique revision identifier that each distributed version control system provides. Access control rules are extended to enable them to require a signature before changes are allowed for inclusion. For example, ACHEL can enforce a company policy that requires changes to be signed off by a manager or an automated validation service before they are deployed.

The remainder of the paper is structured as follows: First, we discuss related work in section 2. Next, we discuss our design and how access control and workflow enforcement is applied. In section 4 we discuss the prototype we have developed, and in section 5 we evaluate our prototype.

2 Related Work

In the state of the art of system configuration tools, various levels of integration with version control systems and granularity of access control are available [10, 14, 18, 19, 24, 32]. But only very limited work on workflow enforcement of changes seems to be available [14, 25]. Using version control in configuration specification repositories is an idea that is used or recommended by most existing tools. Access control on these repositories is usually enforced by authenticating users, and it allows either full access or limited access to directories per user in a manner similar to that of a file system in an operating system. Some tools [16, 18, 24] are able to do a more fine-grained access control on a higher abstraction level, but none of them include any provisions for enforcing a workflow on configuration specification updates.

- BCFG2 [19] includes basic integration with version control repositories through a plug-in that gets the configuration specification from a SVN or Git [3] repository. BCFG2 uses the revision from the repository in reports about the configuration process, but does not provide any access control and relies fully on what the repository provides. Directing Change Using BCFG2 [21] details an approach to deploy complex configuration changes. These changes are split into steps, each of which needs to be deployed before the next can be executed. A script checks the BCFG2 reports for successful deployments before the next step is deployed.

- LCFG [10] does not integrate with version control systems, but the LCFG guide [8] does refer to using a CVS repository as the configuration database.

- Cfengine [13, 14] also recommends using version control systems as a configuration repository. [28] suggests using branches or tags to create different staging environments, for example for testing, production, and development, but no tools seem to be

available to enforce a workflow between these environments.

- Like most other tools, Puppet [25] suggests using version control repositories as a best practice on their wiki. It also contains examples for adding syntax validation before a change is accepted in the version controlled repository [4]. Additionally, the Puppet wiki suggests different branches for different environments, such as testing and production [5, 6].
- DACS [32] includes tight integration with CVS or subversion. It hooks into the version control system to do basic checks such as syntax validation before a change is accepted.
- Devolved Management of Distributed Infrastructure with Quattor [16] describes how several European grid infrastructures manage large distributed infrastructures with sites under different administrative domains. They all use Quattor, a system configuration tool that uses the Pan configuration language [17]. In their workflow they include Subversion as version control repository. One of the problems with their current implementation is the inability to enforce fine-grained authorization. They handle this problem by modularizing the configuration specification using namespaces that the compiler enforces in the file name. This allows Subversion to enforce access control on file names, but the specification in one namespace can still access other namespaces, thus bypassing the Subversion access control.
- Machination [24] provides fine-grained access control based on manipulation primitives of the XML input language. Although at a higher level than providing access based on the file names, there is still an abstraction gap between the configuration specification and the access control. The manipulation primitives express what can be changed in the XML input and do not directly express what can be changed in the input specification, thus causing an abstraction gap between the access control rules and the input specification.
- PoDIM [18] includes rules to filter statements before they are applied to the network. These rules are specified at the same abstraction level as the source and apply directly to the statements in the source specification. However, there are no facilities to enforce a workflow: the specification becomes invalid and cannot be deployed if a change is added that depends on a change that is not approved.

Most tools rely on the coarse-grained access control available in version control repositories. Some tools, such as Machination [24], provide very fine-grained access control based on the configuration specification, but at a lower abstraction level than the specification a sysadmin writes. PoDIM [18] offers filtering of statements at the same abstraction level as the specification but lacks integration with workflow enforcement, thus making it hard to use. Cfengine [14] and Puppet [25] do include provisions to use different branches of version control repository for different stages in deployment in the same configuration server, but cannot enforce workflows between these stages. ACHEL solves these problems by performing access control based on semantically meaningful changes and it adds flexible workflow enforcement.

3 Design of ACHEL

ACHEL provides fine-grained access control which is applied on the semantics of configuration specification changes, as well as version tracking and workflow enforcement. Figure 2 shows a possible workflow and the access rules that can be enforced with ACHEL. This figure also provides an architectural overview of the distributed components of ACHEL. Each agent involved in the configuration of an infrastructure has his own ACHEL repository that is based on a distributed version control repository. The agents are not only system administrators or the system configuration tool, but can also be automated review or other validation services that are required by the company policy for deploying configuration updates. Section 3.1 explains the concepts and the operation of distributed version control systems.

Access control is applied on each of the repositories in Figure 2. Section 3.2 describes how this fine-grained semantic access control is implemented by analyzing the changes between versions at the language level. Finally, section 3.3 details how flexible workflows between the DVCS repositories in Figure 2 are enforced through combining features of distributed version control systems, digital signatures and fine-grained access control.

3.1 Distributed version control

ACHEL builds on a distributed version control system to provide flexible workflows [2] for updating a configuration specification. In contrast with traditional version control systems, where a central repository keeps track of all history, distributed version control systems (DVCS) use a different architecture. Instead of having to interact with a central repository to examine the history, commit changes or use branches, each user has his own local repository. This local repository not only contains a copy of the version he is working on, but also a complete

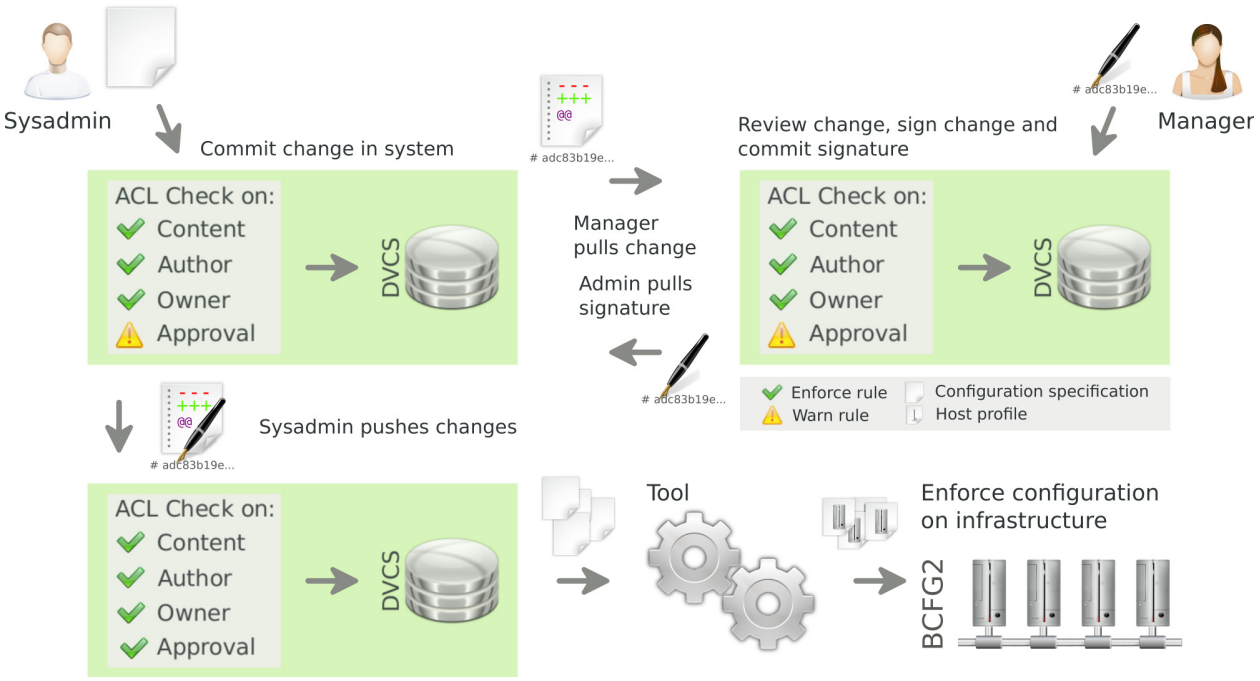


Figure 2: A possible workflow that ACHEL can enforce for the purpose of including a change in the configuration specification repository.

project history, branches, etc. All familiar version control operations such as examining the history, switching to other branches and even committing changes are local operations in a DVCS.

A DVCS enables flexible workflows because it can easily share information such as committed changes or new branches between individual repositories. Information is exchanged via push and pull operations. A push transfers local information to a remote repository, and a pull copies remote information to the local repository. When distributed repositories are used, there is no central repository, so a repository is only authoritative by convention. Another consequence of having distributed repositories is that a DVCS cannot use sequential revision identifiers as traditional version control systems do. Instead, DVCSs use a different mechanism to ensure that revision identifiers are globally unique. However, if two revisions in different repositories have the same history and introduce the same change, then the identifier needs to be equal. For example, Git [3] and Mercurial [27] both use SHA-1 hashes to identify revisions in the repository. The hash is based on the data in the files, on revision metadata and on the parent revisions. One useful consequence of this fact is that a revision identifier identifies and proves the integrity of a revision and all previous revisions. This hash is called the *revision identifier*.

ACHEL provides a sysadmin with flexible development workflows. ACHEL inherits these workflows from the DVCS it is built on. For example, a sysadmin can commit without interfering with changes from others. With authentication and authorization in the mix, flexible workflows become even more important: a user can commit changes that require authorization, without blocking the deployment of other consecutive changes that have already been approved. A DVCS can also be used in a more traditional manner whereby each sysadmin synchronizes his repository with a central *authoritative* repository, which in ACHEL is the repository the system configuration tool uses. In larger federated infrastructures a hierarchy of repositories can be used. Using a DVCS also enables sysadmins to share work with others directly. For example, two sysadmins who are preparing a new configuration for some service, share a common branch and share changes independently of the authoritative repository. Once their work is ready, it can be pushed to the authoritative repository for deployment.

3.2 Access Control

ACHEL enables fine-grained access control based on the semantics of the changes in the configuration specification. In contrast, most existing tools rely on the access control provided by the operating system or version control system. Access rules can be expressed as a function of three things: 1. the contents of a change, using semantically meaningful changes; 2. the owner of the configu-

ration statement that has been changed; 3. the author of the change. To perform access control based on semantically meaningful changes, the access control component of ACHEL needs to understand what the statements in a configuration specification are. The changes to which the access control rules are applied are generated by analyzing the differences between two statements and generating meaningful changes from these differences. In this section the access control approach used is explained; the next section elaborates on the workflow enforcement. The application of access control rules is split up into several steps, as shown in Figure 3.

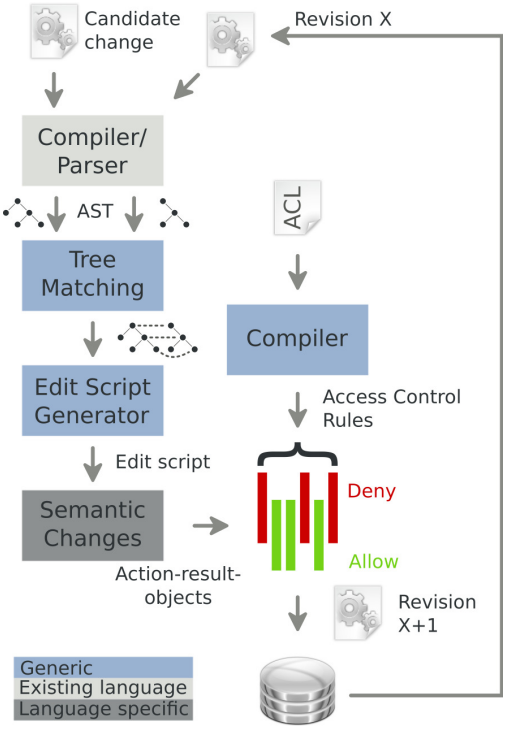


Figure 3: Architecture of the access control component that checks whether a new revision violates any access rules.

All version control systems use diff-like algorithms [30] that operate on flat files to generate changes between two versions of a file. These algorithms create what is called an edit script, which transforms the previous version of a file into the current version. Diff algorithms detect changed lines and create an edit script containing insert and remove line operations. Although this edit script is easy to generate and reapply again, it is impossible to define reasonable access control rules on the insert and remove operations. The abstraction level of these changes is too low because they are expressed in terms of adding and removing lines in a file, while the configuration specification is expressed in configuration related terms. In Listing 3, a diff generated edit script is displayed. The script transforms version 1 of the program in Listing 1 into version 2, which is listed in Listing 2.

Listing 1: Code example: version 1

```
1 var1 = 6
2 var2 = 6
3 prnt(var1 * var2)
```

Listing 2: Code example: version 2

```
1 var1 = 6
2 print(var1 * 7)
```

Listing 3: Edit script between version 1 and 2 generated the diff algorithm

```
1 @@ -1,3 +1,2 @@
2  var1 = 6
3 -var2 = 6
4 -prnt(var1 * var2)
5 +print(var1 * 7)
```

To generate semantically meaningful changes from the configuration specification, ACHEL uses an algorithm that analyses the abstract syntax tree of the configuration specification. System configuration tools build an abstract syntax tree of the configuration specification during compilation. An abstract syntax tree is a tree representation of the abstract syntax [29] of a file. The abstract syntax separates the syntax from the semantics of the specification.
An example of the abstract syntax trees of the program in Listings 1 and 2 is shown in Figure 4. The differences between two versions of a tree are used to generate changes at the right abstraction level, because the abstract syntax tree contains the structure and meaning of each statement, and the composing parts out of which the statement is built. Meaningful Change Detection in Structured Data [15] proposes an algorithm to generate these semantically meaningful changes. Several systems have been developed based on similar algorithms that calculate an edit script from unordered trees. For example, these algorithms analyze changes in source code [23], XML [7, 34], UML [31, 35], and HTML [26].

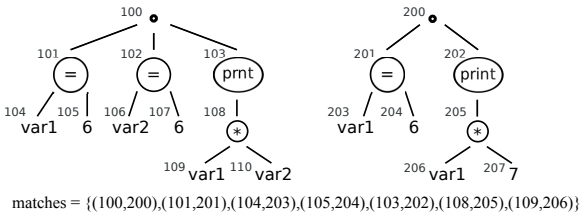


Figure 4: Matching the nodes in the abstract syntax tree of the two versions in Listing 1 and 2.

Listing 4: The edit script for transforming the first tree into the second tree in Figure 4.

```
add    = {node(text="7",parent=205)}
delete = {node(id=102),node(id=106),
          node(id=107),node(id=110)}
update = {node(id=103,text="print")}
```

ACHEL matches the nodes in two abstract syntax trees and generates a tree edit script to transform one tree into the other. It does this by parsing each revision of a configuration specification and generating an abstract syntax tree from it. Each revision of a tree is matched with its previous revision. Figure 4 shows the set (`matches`) of matched nodes obtained by applying algorithm [23] on these two trees. From these matched trees, an edit script is generated that contains a list of add, update and delete node instructions to transform the tree of revision X into the tree of revision $X + 1$. In Listing 4 the edit script of the abstract syntax trees in Figure 4 is shown. The algorithms used to generate an edit script from an abstract syntax tree are language agnostic, as shown in Figure 3.

The approach ACHEL takes until the generation of the edit script is similar to Machination [24]. ACHEL's edit script is comparable to the XML edit instructions of Machination, except that ACHEL can generate an edit script for any arbitrarily complex language. In contrast to Machination, ACHEL translates the edit script into semantically meaningful changes performed on the configuration specification. ACHEL then applies access control on these semantic changes instead of applying them on the edit script, which is of a lower abstraction level. An edit script is expressed in terms of operations on nodes in a tree, while a configuration specification is expressed in configuration related terms. Although an edit script can be generated for any tree, generation of the semantic changes is specific for each language. In section 4 we apply ACHEL's access control on a configuration specification language. The same access control system was used during prototyping to enforce access control on simple configuration files with parameters and sections, sometimes called *.ini* files.

ACHEL allows rules to be specified depending on the current owner of a statement in the configuration specification. Because all operations in an edit script are made by the same user, and this user is known to ACHEL through the DVCS repository, the owner of each statement can be determined. ACHEL determines the owner of each statement by starting at the tree of the first revision and applying the user information and the edit script until the last revision. Every time a node is added or modified, the user that made the change is used as the new owner of that node in the tree. For example, user A creates the first revision of the left tree in Figure 4, so A owns all nodes in that tree. User B makes the changes that result in the right tree in that figure. He removes nodes 102, 106, 107 and 110, updates node 103, and adds node 207. The algorithm will mark B as the owner of the new node and of the updated node 202.

The per node owner information in the abstract syntax tree is not very useful when access control rules are applied on semantic changes. The user information in the abstract syntax tree needs to be mapped onto the generated semantic meaningful changes. The owner information for each node from which a semantic meaningful change is built is used to determine the owner of a meaningful change. For example, in the multiplication statement that consists of node 205, 206 and 207, node 207 is owned by user B in Figure 4, but nodes 205 and 206 are still owned by user A. Depending on the meaning of the statement, ACHEL determines who the owner of the full statement is. In the multiplication from the example, the result of the multiplication will be different because of the change user B made, so he will be the owner of the statement. Because the author of a changeset is used to determine all ownership information, and the ownership and author are used in the access rules, the author information needs to be secure. ACHEL uses digital signatures and a PKI to ensure that all user information in the repository is authenticated.

An important step in developing an access control rule language is identifying the possible statements and their syntax in the configuration language input. Statements include for example assigning a value to a variable, calling a function or creating a new instance of a class or structure. On the statements, access control will be applied. To keep the structure of the needed patterns in the access control language simple, it is important to unify as many statements as possible in order to keep the grammar of the access control language limited. For example, the assign and multiplication statement in Listing 1 and in the abstract syntax tree in Figure 4 can be unified with a `<lhs> <op> <rhs>` structure. In this structure an operation (`op`) is performed on the left-hand side (`lhs`) using the argument on the right-hand side (`rhs`). Changes to a statement are split up into attributes. These attributes are the action performed (add, remove, modify, ...) on the statement, the type of unified statement that it is, and possibly additional attributes of a statement. In our prototype we developed an access control language for the configuration specification language we added to ACHEL.

3.3 Workflow

Each infrastructure has its own policy that defines workflows for deploying changes in the production configuration. For example, changes need to be tested in a test infrastructure before they are deployed in the production

configuration. Current configuration management tools cannot enforce these policies. ACHEL achieves this with a combination of the proposed access control system and the flexible workflows that DVCS's provide. To integrate the two, we extend the access control rules with a clause that requires authorization before a change is allowed.

This new clause specifies the number of authorizations required and a list of users that can authorize a change to be allowed in an ACHEL repository. A change is authorized by digitally signing its unique revision identifier. The identifier is signed with the user's private key. This key is also used by the author to sign information relating to a change. A signature on a revision signifies that a revision is approved by the person or service that is associated with the key used. The reader may recall from section 3.1 that the identifier in a DVCS is a hash that is based on the content and the parent revisions, so a signature also authorizes the full history of the specification.

Authorizing a revision is as simple as adding a signature to the repository metadata and committing the change. This approach generates a new revision in the repository for each new signature. This new revision can be fetched and merged by the user who requested the authorization. When the requesting user has the required number of signatures, he merges his change and the signature revisions into a new revision and pushes it to the repository of the system configuration tool. We rely on existing communication channels such as email or instant messaging for ACHEL notifications. These notifications are required when requests are sent out to authorize changes or to notify other users that a review is finished and a signature is available.

Workflows in ACHEL are based on exchanging changes between the local DVCS repositories that each user controls. ACHEL needs to enforce access control rules on each repository, even though all repositories could possibly have the same changes, because each user has full control over his own repository. This has two important advantages: First, each repository can determine who needs to approve changes before they are accepted. Second, authorization clauses can be set to only warn a user instead of denying access, because a repository revision is required to get a change authorized, and this revision is only available after a change has been included in a repository.

Figure 2 shows an example of a possible workflow. This workflow is similar to the one enforced in the access control rules in Listing 5. In this figure, the manager is a user named Alice, who is a member of the *admin* group in Listing 5. Whenever the sysadmin named Bob makes a change not related to variables named $dhcp\_*$, he needs the authorization of a user in the admin group. The system only warns about the authorization because Bob needs to be able to commit his change, although he does not yet have the authorization. After the change is committed to Bob's local repository, ACHEL enforces the following workflow to include his change in the authoritative repository:

- Bob cannot push his change to the system configuration tool repository, so he emails the users in the admin group that he needs authorization for the change with revision identifier `3d996986778d` in his repository at `http://bugatti:8000`.
- Alice, a user in the admin group, pulls change `3d996986778d` from Bob's repository into her own repository at location `http://ferrari:8000`.
- Alice reviews the change and signs `3d996986778d`, thus creating a new revision `ce5b84ef04a7` in her repository.
- Bob pulls `ce5b84ef04a7` from Alice's repository at `http://ferrari:8000` into his own repository.
- Bob creates a new merge revision `acdd701f412c` that now satisfies all access rules.
- Bob pushes `acdd701f412c` to the system configuration tool repository for the purpose of scheduling it for deployment onto the infrastructure.

Whenever a user pushes his specification changesets to another repository, other changesets that possibly conflict could have been included. When this occurs, these changesets need to be merged. During merging, two scenarios can occur. These scenarios are illustrated in Figure 5. If the changesets do not conflict with each other, a merge changeset can be created that does not introduce any additional changes. If all changesets prior to the merge satisfy all access rules, then the merge changeset can be accepted by ACHEL. Whether this changeset should be accepted is highly dependent on the configuration specification language to which ACHEL is applied. If the configuration language is fully declarative, then the order of configuration statements does not matter and changes can be applied in any order, as long as they do not conflict. If the order in which changes are applied does matter, then a special merge permission should be introduced so that only users with this permission can merge changesets.
In a second scenario some changesets can conflict, so the merge changeset needs to include additional changes to resolve the conflicting changesets. If this occurs, then these new changes need to satisfy all access control rules, just like any other normal change. Such a merge changeset can also include non-conflicting changesets, in which case the same restrictions apply as in the other scenario.

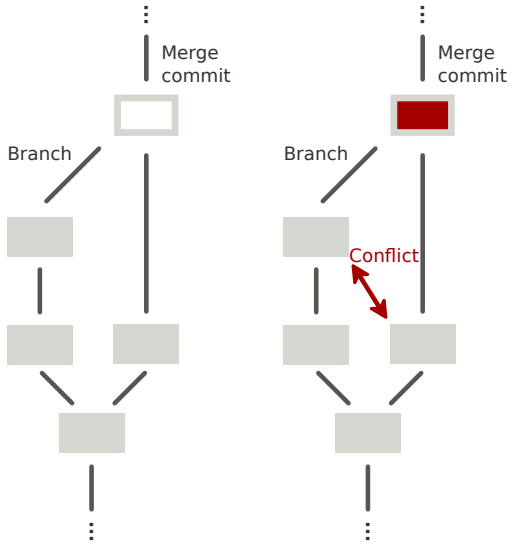


Figure 5: Merging branches of a configuration specification repository. Left: a branch is merged without any conflicts. Right: two changesets conflict and the merge changeset contains additional changes to resolve the conflict.

Conflicting changesets should not occur very often in a real infrastructure. Because responsibilities are mostly non-overlapping, modularizing the configuration specification in files that do not contain overlapping responsibilities will prevent conflicting changes, insofar as this is possible.

4 Prototype

This section describes a prototype configuration language and compiler: Westmalle. Westmalle is a simple configuration language that we extended with ACHEL support.

The Westmalle configuration language design is inspired by the functionality provided by LCFG2. The Westmalle configuration language is a declarative language with only three operations. It can *import* configuration directives from libraries, it can *add* values to lists and it can *assign* values to a variable name. Because of its declarative nature, variables can only be assigned once. Various values can be assigned to these variables, including string literals, functions that can interface with other systems such as template systems, and structures with named attributes.

Structures are an important part of Westmalle. Each structure has a list of named attributes. Structures are not declared and are created in the same way as classes are instantiated in languages such as Python. Attributes are added every time a value is assigned to an unknown attribute. Westmalle does not type check these structures, but therein lies its strength. Because classes are easily created, a user can create specifications that match what, for example, LCFG2 or BCFG2 require. When the compiler finishes resolving all variables in the specification, then these structures and their attributes are used to generate the output. The compiler can be used in two ways:

1. As a BCFG2 plug-in that exposes the return values of the BCFG2 client probes [20] as variables in the configuration language. These probes are used in BCFG2 to find information about the managed system.

2. Generating XML output that conforms to the structure proposed in Configuration tools: Working together [12].

We added support for ACHEL to Westmalle. The architecture of our prototype is shown in Figure 6. The right box is the Westmalle compiler, the left box is the ACHEL support infrastructure. The ACHEL support infrastructure contains a distributed version control repository at the bottom and a set of allow/deny access control rules. If a user wants to push a change to the repository, that change has to pass the access control rules before it is committed to the repository. Once the change is committed, the Westmalle compiler generates XML or BCFG2 specifications, which in turn can be used by a deployment engine to enforce the specification.

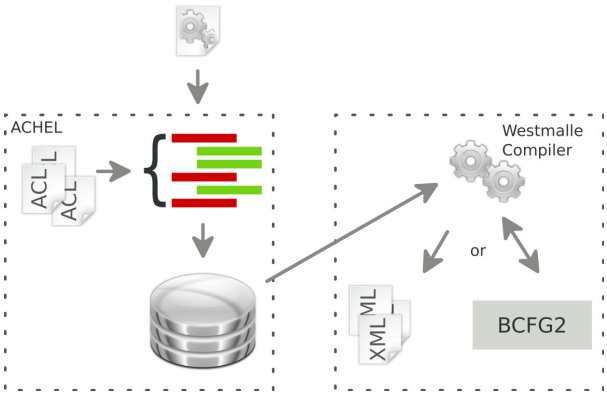


Figure 6: An architectural overview of our ACHEL prototype.

The access control rule enforcement is implement as show in Figure 4. The access control language defines pattern matching rules that match the attributes generated from the semantic changes in the configuration specification. Each rule contains an action that is taken if the patterns from the rule match the attributes of a change. Access control rules are evaluated in the order they are declared in the source file. When a rule matches, evaluation stops and the action of that rule is taken.

The rules in the access control language consist of two different parts. The first is a generic part that specifies the author, the owner, the required signatures and the action taken when a rule matches. The second part is language-specific and contains the structure the semantically meaningful edit instruction needs to match before an action will be executed. In the access control language, user groups can be defined and each user can be included in multiple groups.

Listing 5: Access control rules and group definitions

```
1  # list of senior admins
2  define admins as
       admin1@cs.kuleuven.be,
       admin2@cs.kuleuven.be
3
4  # allow everyone to create dhcp
5  # configuration
6  allow to:
7      add import dhcp.*
8      * assign * to dhcp_*
9
10 # senior admins can do anything
11 allow admins to:
12     *   *
13
14 # others can do anything if
15 # approved by a senior admin
16 allow to:
17     *   *
18     authorised by 1 admins
```

Listing 5 shows an example of possible access control rules.

- Line 2 defines a group of users. This group can be used in the access control rules.
- Line 6 starts a new rule with two language specific rules on the next two lines. The rule allows all changes that match one of the language specific sub-rules on the next lines. The first sub-rule on line 7 matches all changes that *add* an *import* statement. This import statement is constrained to libraries matching the wildcard string *dhcp.**. The second sub-rule on line 8 matches changes that *assign* any value (hence the second wildcard) to a variable that matches the wildcard string *dhcp_**. The first wildcard of this sub-rule signifies that these changes can be *added, modified or removed*.
- The rule on line 11 allows changes that match any of the sub-rules, if the author of the change is a user from the admin *group*. The sub-rule on line 12 matches every change.
- The rule on line 16 allows any change (the two wildcards on line 17) by any author, if it is authorized by one or more users from the administrator group (line 18).

A set of rules such as in Listing 5 starts with a generic header that expresses the action taken when a rule matches and lists the possible authors of the change. In the prototype, actions are limited to *deny* or *allow* the change. The action of the first matching rule is used and no other access control rules are checked. After the action, an optional list of authors of a changeset can be given, followed by the *to* keyword, a colon and a newline. The list of users is a comma separated list of email addresses or group names to allow role based access control.

The body of the rule can contain multiple directives that are divided into generic and language specific lines. The generic lines start with the *owned by* and *authorized by* keywords. The *owned by* clause specifies a list of owners of the original statement required for this rule to match. This list is identical to the author list described in the previous paragraph. The *authorized by* clause is followed by an optional number and a list of users. The number indicates how many signatures are required for a rule to match. If no number is provided, the number one is assumed.

The other rules in the body are language specific rules. These rules start with *add, modify or remove*, followed by a pattern to match a meaningful change. The first keyword specifies the change made in the changeset required for this rule to match. If multiple language specific rules are present in the body of a rule, they are treated as separate rules with a common generic part. For example, in Listing 5 for the rule on line 6, two rules with the same header are created because of the two language specific rules on the next lines. For each different structure in the configuration language, a different language specific rule syntax is required. The number of structures in a language depends on how similar statements in the configuration language are. Each rule syntax consists of matching patterns that are applied to the matching attributes of a semantic change. Each attribute is matched with a literal value, a string that may contain wildcards, or a regular expression.

For the simple configuration language we developed, we are able to unify all statements within the same structure. This means that we can match all statements in the configuration specification with the same rule syntax. Rule syntax contains at most three patterns to match the three attributes of each change to the configuration specification. These attributes are: the operation, the right-hand

side, and the left-hand side. The most important attribute of the configuration language is the *operation* of a statement. In the prototype there are only three available operations: import, assign and add. After the operation, a rule can also contain an optional right-hand side, as well as an optional left-hand side pattern.

ACHEL uses hg (short for Mercurial [27]) as the version control system in our prototype. It is a lightweight DVCS written in Python. It has been adopted by very big software projects such as OpenJDK and Mozilla. Like other well known DVCS control systems such as Git [3] and BitKeeper [1], hg uses content hashes as identifiers. Mercurial was selected for our ACHEL prototype because it is open source, it is written in Python and it has extensions for signing revisions.

ACHEL can merge changesets. A merge is handled differently depending on the scenario:

- If changesets have been merged without conflict and the merged branches introduce changes in different files, no additional permissions are required.
- If changes are introduced in the same file, a user that commits the merge changeset needs merge permissions. Merge permissions can be dependant on authorization, so they can be forced to go through a review process.
- If changesets conflict, then the new changes that solve the conflict need to satisfy the access control rules. The two previous rules apply to any non-conflicting changes that are also merged.

Digital signatures are used to verify the users in the author, owner, and authorization constructions in the access control rules. In the prototype, GPG is used to generate these digital signatures. The email addresses linked with the private key that is used to create digital signatures are used in the access control rules to identify users. GPG was chosen over x509 certificates for two reasons: First, because Mercurial already has support for signing revisions with GPG signatures. Second, because it is very lightweight and works well in a federated environment. To establish trust, GPG can be configured to only trust signatures from keys within a certain trust level.

ACHEL can support other DVCS and infrastructures for digital signatures. It uses the version control repository to report the email addresses of the users that created or signed a changeset and it relies on the ability of the repository to verify the user information on the basis of digital signatures. This way there is no direct coupling between the PKI used and ACHEL. Because the interface between ACHEL and the DVCS is small, new distributed version control systems are easy to add.

5 Evaluation

In this section we evaluate ACHEL in two case studies. The first case validates the improvements achieved in ACHEL for environments where several administrators manage the same infrastructure but are responsible for different aspects of the infrastructure. This case focuses mainly on the access control features with limited workflow enforcement. The second case validates the use of ACHEL in federated infrastructures with a focus on workflow, such as used in grid computing.

5.1 Case 1

This first case validates the prototype in an infrastructure managed by several system administrators with varying levels of seniority, each of whom has his own responsibilities. The update policy of this infrastructure is:

- Sysadmins can only make changes to the aspects of the infrastructure they are responsible for.
- Everyone can make any change if it is authorized by a senior sysadmin.
- Senior sysadmins can change anything.

The access control rules use group names to identify users, so users can be assigned according to their responsibilities. In Listing 6, an excerpt with access control rules is shown. The first rule on line 2 forces every change to encode the type in the variable name. For example, the variable with the configuration file for the dhcp server should be called `net_file_dhcpd_conf`. Six rules are declared for each type that can be used with the BCFG2 plug-in. If a variable does not match this convention, the change will be denied. The second rule on line 11 stipulates that every user in the *senioradmin* group can do anything. Statements in a change will only get to this rule if they conform to the previous rule. The rule on line 15 authorizes any change from any user if it has been approved by a user from the senioradmin group.

The next two rules are specific for users in the *netadmins* group. These users are only allowed to change the network configuration related to files located in `/etc/network` and services called `network` and `dhcpd`. The rule on line 26 allows a netadmin to import dhcp configuration libraries, to add entries to the global list of dhcp clients and to declare variables that are prefixed with `net_`. The rule on line 20 limits this to the files and services that are allowed. If none of the rules above matches, the change is denied.

Listing 7 shows a possible configuration an admin responsible for the network configuration has written in our configuration language. This code example creates a configuration file called

Listing 6: Case 1: Access control rules and group definitions

```
# enforce some conventions on everyone
deny to:
  * assign  File()          to  /^[^_]+_(?!file_)[\S]+$/
  * assign  Package()       to  /^[^_]+_(?!pkg_)[\S]+$/
  * assign  Service()       to  /^[^_]+_(?!service_)[\S]+$/
  * assign  Directory()     to  /^[^_]+_(?!dir_)[\S]+$/
  * assign  Symlink()       to  /^[^_]+_(?!ln_)[\S]+$/
  * assign  Permissions()   to  /^[^_]+_(?!perm_)[\S]+$/

# senior admins can do anything else
allow senioradmin to:
  * * *

# allow admins to do everything if a senior admins approves
allow to:
  * * *
  authorised by 1 senioradmin

# network related configuration
deny netadmins to:
  # deny files other then those in /etc/network
  * assign /^(?!\/etc\/network\/)\S+/ to /^net_file_\w+\.name$/
  # deny services other then dhcpd and network
  * assign /^(?!(dhcpd$|network$))\w+$/ to /^net_service_\w+\.name$/

allow netadmins to:
  * import  /^dhcp/
  # allow adding a list of values to the net_dhcp_clients list
  * add     /^\[[^\]]$/ to  /^net_dhcp_clients$/
  # allow only variables prefixed with net (ignore rhs)
  * assign  *           to  /^(?!net_)\S+$/
```

`/etc/network/interfaces` and enables the `network` service. This is allowed by the access control rules defined in Listing 6. The code starting from line 17 creates a `/etc/hosts` file. To do this, a user needs to be either a senior admin himself or else he needs to have the permissions of a senior admin.

A network administrator named Kris, who is not a senior administrator, wants to include his configuration specification in the main repository. To do this he needs permission from a senior administrator named Jean. Kris commits the change to his ACHEL repository and receives a warning that he needs permission from a member of the senioradmin group to satisfy the access control rules enforced on the main repository. Kris emails Jean and asks him to review the change with identifier `c8d8c7780069` in Kris' repository, which is available at `http://alpha:8000`. Jean pulls this change from Kris' repository and ACHEL warns Jean that the change needs to be authorized. Jean reviews the change, signs it and commits the signature to his repository. Jean then emails Kris that he approves the change and that Kris can pull the signature with identifier `3a526359364e` from his repository, which is available at `http://beta:8000`.

Kris pulls Jean's signature into his repository and ACHEL now shows that all changes satisfy the access control rules. Kris can now push his change and Jean's signature into the main repository in order to deploy it on the infrastructure.

5.2 Case 2

Quattor is used in federated infrastructures composed of multiple physical sites. Devolved Management of Distributed Infrastructures [16] explains how Quattor is used in a few different federated infrastructures. In this case we apply ACHEL to the BEGrid infrastructures described in [16]. BEGrid uses a model of highly-autonomous sites that loosely collaborate. Each site has its own configuration servers, but it gets its configuration from a central Subversion repository. In this repository, both common and site-specific configuration specifications are stored.

In this case we will focus on the application of ACHEL's workflow enforcement features on a BEGrid-like infrastructure. Each site configuration server uses its own repository instead of getting its configuration specification from the central repository. Each site has its own authoritative ACHEL repository from which the site configuration server gets its specification. The institution that coordinates the grid also maintains a repository from which each site updates their common configuration specification. In Figure 7 the flow of changes between repositories is shown. The workflow between the repositories in Figure 7, is already supported by any normal DVCS.

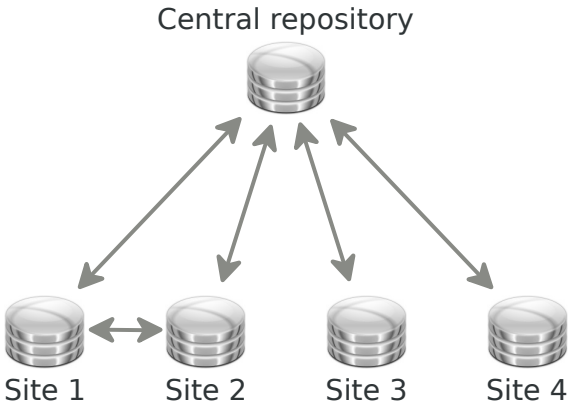


Figure 7: BEGrid repositories using ACHEL

In this case we will enforce policy rules on this workflow using ACHEL:

- Every site has its own ACHEL repository that is used by the site's configuration server.
- System administrators commit to the site repository, possibly with extra policy rules specific for their own site.
- Sysadmins can only change the configuration related to their own site.
- Changes by sysadmins from other sites have to be approved by a manager of the affected site.
- Changes to the common templates have to be approved by at least three out of five managers.

Our configuration language does not match the PAN [17] language used by Quattor in BEGrid, but this is not an issue for demonstrating the workflow capabilities of ACHEL. For the access control rules in Listing 8, we assume that the site-specific and common configuration can be identified by the first word in the variable name: `common_` or `site_`. On lines 1-5 of the listing groups are defined containing the sysadmins of each site and all the managers. These groups are used in the access control rules we describe next.

The rules in Listing 8 implement the policy we described earlier in this section. The first rule on line 7 stipulates that changes to the *common* configuration need authorization from at least three users from the management group. Lines 11-16 define the access rules for the configuration of site 1. The first rule limits access to users in the site1 group, and the second rule stipulates that other users have access to site 1 configuration only if they have authorization from the site1 manager. The

Listing 7: Case 1: Network configuration specification

```
1  import base
2
3  # configure network interfaces
4  net_file_interfaces = File()
5  net_file_interfaces.name = "/etc/network/interfaces"
6  net_file_interfaces.owner = "root"
7  net_file_interfaces.group = "root"
8  net_file_interfaces.perms = "0644"
9  net_file_interfaces.content = source("net/interfaces.$hostname")
10
11 # network service needs to be enabled
12 net_service_network = Service()
13 net_service_network.name = "network"
14 net_service_network.status = "on"
15
16 # use template for /etc/hosts with loopback and host ip
17 net_file_hosts = File()
18 net_file_hosts.name = "/etc/hosts"
19 net_file_hosts.owner = "root"
20 net_file_hosts.group = "root"
21 net_file_hosts.perms = "0644"
22 net_file_hosts.content = template("net/hosts.tmpl")
```

rules on lines 18-37 provide for similar rules for the other three sites. Site 1 and site 2 are developing a new common feature defined under `common_new`. The last rule on line 39 allows users of both sites to work on it. This rule stipulates that users from the groups site 1 and site 2 have access to all configurations under `common_new`.

Each repository can have its own access control rules in ACHEL. For example, the last rule on line 39 only makes sense in the repositories of sites 1 and 2. This also holds for other the site-specific rules, which only need to exist at the site itself and at the main repository.

5.3 Limitations and Future Work

In this paper we have prototyped ACHEL on a simple configuration language. One area for future work involves the need to add ACHEL support to existing (more complex) configuration languages. Another area involves the need to improve the usability of ACHEL by adding support for processing authorization requests. A third area involves the need to provide support for meta-ACL's: i.e. to provide access control rules for specifying the access control rules.

Supporting existing configuration languagues

ACHEL itself is not a product, it is a generic framework that can be reused in existing configuration languages. The framework offers support for meaningful change detection, and for the enforcement of access control rules and workflow. To add support for ACHEL to an existing configuration tool, it must:

1. either add access control constructs to its language or use a separate access control language;
2. and provide ACHEL with an abstract syntax of its configuration specification.

The complexity of an access control language is directly linked with the number of different semantically meaningful change structures that need to be matched. The reader will recall that in our prototype we were able to unify all three language constructs within a single structure, but this will no longer be possible for more complex configuration languages. The expressions in Listing 6 are very powerful because they use regular expressions, but they are complex to use. If ACHEL were to be applied to more complex configuration languages such as Cfengine [14] or Puppet [25], the current access control language and matching model would probably become needlessly complex. These languages are more expressive and contain more than one structure that needs to be matched.

To support more complex languages and make the access control language easier, some enhancements are required. The first enhancement would be to use the type system of the configuration language to enforce rules, so that types and namespaces do not need to be encoded in the names of the variables as in the first case. Another required enhancement of the access control lan-

Listing 8: Case 2: BEGrid example access control rules

```
1  define management as
       director@begrid.be,
       manager@site1.begrid.be,
       manager@site2.begrid.be,
       manager@site3.begrid.be,
       manager@site4.begrid.be
2  define site1 as ...
3  define site2 as ...
4  define site3 as ...
5  define site4 as ...
6
7  allow to:
8    authorised by 3 management
9    * * * to /^common_/
10
11 allow site1 to:
12   * * * to /^site1_/
13
14 allow to:
15   authorised by
         manager@site1.begrid.be
16   * * * to /^site1_/
17
18 allow site2 to:
19   * * * to /^site2_/
20
21 allow to:
22   authorised by
         manager@site2.begrid.be
23   * * * to /^site2_/
24
25 allow site3 to:
26   * * * to /^site3_/
27
28 allow to:
29   authorised by
         manager@site3.begrid.be
30   * * * to /^site3_/
31
32 allow site4 to:
33   * * * to /^site4_/
34
35 allow to:
36   authorised by
         manager@site4.begrid.be
37   * * * to /^site4_/
38
39 allow site1, site2 to:
40   * * * to /^common_new_/
```

guage would be to include a mechanism for abstracting certain details: for example, a mechanism for using templates to match certain structures and to hide the complexity of the regular expressions used. A final enhancement would make it possible to combine rules using different operators.

The second requirement for ACHEL integration is that ACHEL should be provided with an abstract syntax tree. If a configuration tool includes a formal grammar definition of its language, this grammar can be reused. Another option is to dump the internal abstract syntax tree structures of a system configuration tool.

Supporting authorization request processing

Currently, ACHEL relies on existing communication for notifications related to authorization requests and signatures. To improve the usability of ACHEL in real infrastructures, a plug-in or tool is needed for automatically processing authorization requests by pulling the change, requesting a signature from the user, committing the signature and notifying the author of the signed change. ACHEL should also support multiple DVCS's and digital signatures for real world deployment in order to match existing practices and tools in an infrastructure. Bugtracker tools could also be useful in real world applications, because some bugtrackers include DVCS integration.

Meta-ACL's

Our prototype does not contain support to enforce access control on the access control language itself. The access control available in the underlying distributed version control system can in most cases be reused, because typically only a few users in an infrastructure are allowed to define policy related rules. Technically it is possible to apply the same change detection approach to the access control language itself and provide a meta access control language. This would also make it possible to implement a mechanism for delegating permissions.

6 Conclusion

ACHEL provides integration of fine-grained access control with existing configuration tools and it can enforce configuration change workflows in federated infrastructures. Moreover, ACHEL combines these capabilities with distributed version tracking and cryptographic secure authentication. It also makes access control rules easier to write because they are defined at the same abstraction level as the configuration specification language. And finally, because large parts of ACHEL are

language agnostic, support for new configuration languages can be added with minimal effort.

7 Acknowledgments

We would like to thank Wouter De Borger and Stefan Walraven for proofreading this paper. We also thank Mark D. Roth for his work on shepherding this paper this paper and the anonymous reviewers for their valuable feedback.

References

[1] BitKeeper Website. http://www.bitkeeper.com, 2007.

[2] Bazaar version control: Workflows. http://bazaar-vcs.org/Workflows, 2008.

[3] Git Website. http://git-scm.com/, 2009.

[4] Keep your Puppet manifests under version control. http://reductivelabs.com/trac/puppet/wiki/VersionControlPuppet, 2009.

[5] Puppet Change Management. http://reductivelabs.com/trac/puppet/wiki/ChangeManagement, 2009.

[6] Using Multiple Environments in Puppet. http://reductivelabs.com/trac/puppet/wiki/UsingMultipleEnvironments, 2009.

[7] AL-EKRAM, R., ADMA, A., AND BAYSAL, O. diffX: an algorithm to detect changes in multi-version XML documents. In *CASCON 05: Proceedings of the 2005 Conference of the Centre for Advanced Studies on Collaborative Research* (2005), IBM Press, pp. 1–11.

[8] ANDERSON, P. *The Complete Guide to LCFG*, 2003.

[9] ANDERSON, P. *Short Topics in System Administration 14: System Configuration*. Berkeley, CA, 2006.

[10] ANDERSON, P. LCFG: A large scale UNIX configuration system. http://www.lcfg.org, 2008.

[11] ANDERSON, P., AND COUCH, A. What is this thing called System Configuration? *LISA Invited Talk* (November 2004).

[12] ANDERSON, P., AND SMITH, E. Configuration tools: Working together. In *Proceedings of the 19th Large Installations Systems Administration (LISA) Conference* (Berkeley, CA, USA, 2005), USENIX Association, pp. 31–38.

[13] BURGESS, M. Cfengine: a site configuration engine. *USENIX Computing Systems 8*, 3 (1995), 309–402.

[14] BURGESS, M. Cfengine Website. http://www.cfengine.org, 2009.

[15] CHAWATHE, S. S., AND GARCIA-MOLINA, H. Meaningful change detection in structured data. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data - SIGMOD 97 SIGMOD 97* (New York, NY, USA, 1997), ACM, pp. 26–37.

[16] CHILDS, S., POLEGGI, M. E., LOOMIS, C., MEJAS, L. F. M., JOUVIN, M., STARINK, R., DE WEIRDT, S., AND MELI, G. C. Devolved Management of Distributed Infrastructures With Quattor. In *Proceedings of the 22nd Large Installation System Administration (LISA) Conference* (Berkeley, CA, USA, 2008), USENIX Association, p. 175189.

[17] CONS, L., AND POZNANSKI, P. Pan: A high-level configuration language. In *Proceedings of the 16th USENIX Conference on System Administration (LISA)* (Berkeley, CA, USA, 2002), USENIX Association, pp. 83–98.

[18] DELAET, T., AND JOOSEN, W. PoDIM: A language for high–level configuration management. In *Proceedings of the 21st Large Installation System Administration (LISA) Conference* (Berkeley, CA, USA, 2007), USENIX Association, pp. 1–13.

[19] DESAI, N. Bcfg2: A Pay as You Go Approach to Configuration Complexity.

[20] DESAI, N., BRADSHAW, R., AND HAGEDORN, J. *Bcfg2 Manual*, July 2006.

[21] DESAI, N., BRADSHAW, R., HAGEDORN, J., AND LUENINGHOENER, C. Directing change using Bcfg2. In *Proceedings of the 20th Large Installation System Administration (LISA) Conference* (Berkeley, CA, USA, 2006), USENIX Association, pp. 215–220.

[22] DESAI, N., BRADSHAW, R., MATOTT, S., BITTNER, S., COGHLAN, S., EVARD, R., LUENINGHOENER, C., LEGGETT, T., NAVARRO, J.-P., RACKOW, G., STACEY, C., AND STACEY, T. A case study in configuration management tool deployment. In *Proceedings of the 19th Large Installation System Administration (LISA) Conference* (Berkeley, CA, USA, 2005), USENIX Association, pp. 39–46.

[23] FLURI, B., WUERSCH, M., PINZGER, M., AND GALL, H. Change Distilling: Tree Differencing for Fine-Grained Source Code Change Extraction. *IEEE Transactions on Software Engineering 33*, 11 (2007), 725–743.

[24] HIGGS, C. Authorisation and Delegation in the Machination Configuration System. In *Proceedings of the 22nd Large Installation System Administration (LISA) Conference* (Berkeley, CA, USA, 2008), USENIX Association, pp. 191–199.

[25] KANIES, L. Puppet Website. http://reductivelabs.com/projects/puppet/, 2008.

[26] LIM, S.-J., AND NG, Y.-K. An Automated Change-Detection Algorithm for HTML Documents Based on Semantic Hierarchies. *Data Engineering 0* (2001).

[27] MACKALL, M. Towards a Better SCM: Revlog and Mercurial.

[28] MATES, J. Storing CFEngine configuration in CVS. http://sial.org/howto/cfengine/repository/, 2009.

[29] MCCARTHY, J. Towards a mathematical science of computation. *Information Processing 62* (1962), 21–28.

[30] MYERS, E. W. An O(ND) difference algorithm and its variations. *Algorithmica 1*, 1 (1986), 251–266.

[31] OHST, D., WELLE, M., AND KELTER, U. Differences between versions of UML diagrams. In *Proceedings of the 9th European Software Engineering Conference, held jointly with 10th ACM SIGSOFT International Symposium on Foundations of Software Engineering - ESEC/FSE 03 ESEC/FSE 03* (New York, NY, USA, 2003), ACM, pp. 227–236.

[32] ROUILLARD, J. Distribution and Configuration System. http://www.cs.umb.edu/~rouilj/DACS/, 2009.

[33] TRAUGOTT, S., AND HUDDLESTON, J. Bootstrapping an Infrastructure. In *Proceedings of the 12th USENIX Conference on System Administration (LISA)* (Berkeley, CA, USA, 1998), USENIX Association, pp. 181–196.

[34] WANG, Y. X-Diff: an effective change detection algorithm for XML documents. In *Proceedings 19th International Conference on Data Engineering (Cat No 03CH37405) ICDE-03* (Los Alamitos, CA, USA, 2003), vol. 0, IEEE Computer Society, p. 519.

[35] XING, Z., AND STROULIA, E. UMLDiff: an algorithm for object-oriented design differencing. In *ASE 05: Proceedings of the 20th IEEE/ACM international Conference on Automated Software Engineering* (New York, NY, USA, 2005), ACM, pp. 54–65.

CIMDIFF: Advanced difference tracking tool for CIM compliant devices

Ramani Routray
IBM Almaden Research Center
routrayr@us.ibm.com

Shripad Nadgowda
IBM India Systems and Technology Lab.
shripad.nadgowda@in.ibm.com

Abstract

Total Cost of Ownership (TCO) for any enterprise scale data center is significantly dependent upon the effectiveness of the system management solutions and procedures deployed. Complexity of managing a data center increases as various enterprise applications demand diverse sets of requirements, leading to a very heterogeneous environment often fueled by diverse emerging technologies. Emergence of the industry standard Common Information Model (CIM) has introduced uniformity and interoperability into this complex managed environment.

In this paper, we describe a tool CIMDIFF that provides syntactic and semantic difference tracking for CIM compliant devices in both spatial and temporal flavors. Since this problem is NP-hard, in this paper we present an efficient technique that combines domain specific object oriented knowledge with hierarchical structure of CIM-XML to derive meaningful differences. We demonstrate the value of this tool for a) tracking difference in device characteristics b) verification of proper operation as well as automated validation of management software given the limited resources of testing infrastructure. An experimental evaluation of this tool in a complex data center is provided.

1 Introduction

Any modern day enterprise-scale data center is heterogeneous in nature. Varieties of Service Level Agreements (SLAs) mandated by wide range of deployed applications along with acquisition of emerging technologies drive the heterogeneity . System Management solution(s) deployed to manage the data center provide the basic tuning knobs to adjust in order to meet the SLA goals in terms of reliability, availability, performance etc.. With the advent of virtualization, ensuring the performance guarantees, security, and fault isolation have become even more challenging. E.g. to isolate an application bottleneck; configuration, performance metrics have to be collected across server(s), network element(s) and storage controller(s). These metrics have to be investigated after correlating across physical and virtual layers. Whether, its the internal data center of an enterprise or the service offerings such as cloud computing [1, 2], *System Management* is the key. To enable seamless management in the complex and heterogeneous environments, open industry standards are necessary. Standards like Common Information Model (CIM) [3] from Distributed Management Task Force (DMTF) [4], Storage Management Initiative Specification (SMI-S) [5] from Storage Networking Industry Association (SNIA) [6], Simple Network Management Protocol (SNMP) [8], WBEM [9], SMASH [7], WMI [10] etc.. have provided excellent base to ensure interoperability between a wide array of multi-vendor data center elements, including fiber channel and IP networking components, storage components, servers, operating systems, software infrastructure and applications. System management and storage management solutions like IBM TPC [20], IBM Director [21], EMC Control Center [13], HP Systems Insight Manager [19], Microsoft Systems Center [22] have demonstrated the use of these standards to bring unified interoperable open management to complex heterogeneous managed environment.

CIM provides a common extensible base definition of management information for systems, networks, applications and services. CIM's common definitions enable vendors to exchange semantically rich management information between systems over the network. Vendor devices expose the management information through a software module called *CIM Agent*. CIM agent is either embedded in the device hardware or externally installed and configured to point to the managed device to report the management information. One CIM agent can report management information of multiple devices based on the configuration settings. CIM agents may be automatically discovered using Service Location Protocol

(SLP) or are explicitly specified by the system administrator. CIM agent software is a set of software modules called CIM Providers that are plugged into the Common Information Model Object Manager (CIMOM). CIM Providers are like servlets, that are plugged into the CIMOMs which are like the application servers. Open source CIMOMs [16, 17, 18] available are commonly used in the industry. In this paper, we refer the complete managed device information reported by CIM Agent as *CIM Repository*. This information is either cached in an internal format in the device CIM agent or retrieved on-demand through device instrumentation. Standard CIM Client [15] is used to query information from the CIM agent or to invoke configuration change operations. Adapters are also available to map SNMP to CIM Object, which is returned to the management applications. Information exchanged over network conforms to CIM-XML format. Since XML is hierarchically structured, difference tracking in such format is known to be NP-hard due to the nesting.

In this paper, we propose a tool CIMDIFF that can efficiently track syntactic and semantic difference across CIM repositories in both spatial and temporal favors. Using this tool, user/administrator can get answers to questions of following nature: a)Spatial Difference: What are the difference in characteristics between the two storage subsystems in my data center such as capacity, available space, number of disk drives, number of fiber channel ports etc..? What are the difference in configuration characteristics across two virtual machines (VMs) ? b)Temporal Difference: What are the configuration changes to the storage subsystem from past one week such as volumes created/deleted/modified, volumes assigned/unassigned etc..? What are zone configuration changes to the fiber channel fabric in last one week ? This tool uses an interesting hashing technique combined with KnowledgeBase of standard recipes that helps track differences at i) CIM construct level ii) device characteristics level to track the device configuration changes. Semantic difference provides deep insight to administrators about the data center. Syntactic difference has proved to be very useful in real production testing environment of storage/system management suite. CIMDIFF can also be used to aid conformance testing [23] by tracking effect of CIM Agent version changes when vendors update versions of their CIM Providers.

2 Background

CIM Repository reported by a CIM agent contains the complete management information of device(s). For example, a storage subsystem CIM Repository would contain information regarding the storage subsystem, storage pools, storage volumes, storage pool to storage volume association, fiber channel ports, masking/mapping information etc.. CIM Agent also reports the storage volume performance and fiber channel port performance statistics on demand. Sample CIM-XML information stream reported for a IBM DS6000 Storage subsystem is represented in Figure 1.

2.1 Model and Problem Definition

The CIM Repository (C_r) is a collection of CIM Instances (C_i) that are instances of CIM Class(es) (C_c). The structure of a CIM Instance is shown in Figure 1. Each CIM Class (C_c) defines the structure of either an entity class or an association class. An example of an entity class is `IBMTSDS-ExtentPool` that represents a Storage Pool in a IBM Storage Subsystem. Following the inheritance model of CIM, `IBMTSDS-ExtentPool` class extends from standard CIM defined class `CIM-StoragePool`. Similarly, entity class `IBMTSDS-Volume` represents a Storage Volume that extends from CIM defined `CIM-StorageVolume`. Example of an association class is `IBMTSDS-AllocatedFromExtentPool` that represents the association between Storage Pool(s) and Storage Volume(s) extending from standard CIM class `CIM-AllocatedFromStoragePool`. Single CIM Repository can contain CIM Instances of one device or multiple devices depending upon the number of devices attached to the CIM agent.

We denote a tree T by its nodes N. In the context of this paper, CIM Repository (C_r) is a tree. Children of a node are represented as $n \in N$. A CIM Instance is composed of one CIM Objectpath and zero or more CIM Properties. Node N can be a CIM Instance (C_i) or a CIMObjectpath(C_{op}) or a CIM Property(C_p). Edit operation e applied to original tree T_1 transforms the tree into T_2 is described as $T_1 \overrightarrow{e} T_2$. CIMDIFF computes *minimum-cost optimal edit script*, a sequence of basic edit operations that depicts the transformation $T_1 \overrightarrow{e} T_2$. Definitions are formally represented as:

$$C_r = \{\ N^+\ \}$$
$$N = \{\ C_i | C_{op} | C_p\ \}$$
$$C_i = \{\ C_{op}, C_p^*\ \}$$
$$C_{op} = \{\ C_p^+\ \}$$

Edit Operations: Three edit operations (e) that are evaluated for computing difference between two CIM repositories are:

- **Insert:** Insert operation creates a new node N in the tree T. Nodes N that are CIM Instance (C_i) qualify as an insert in a CIM Repository (C_r).

Figure 1: Structure of CIMInstance

- **Delete:** Delete operation is the removal of a node N in the tree T. Node N that are CIM Instance (C_i) qualify as a delete in a CIM Repository (C_r).

- **Update:** Update operation is the update of a node N in the tree T. Node N that are CIM Instance (C_i) qualify as an update in a CIM Repository (C_r), only if the CIMObjectpath(C_{op}) is unchanged across original and modified versions. Insertion, deletion or modification of one or more CIM Property(C_p) that are non-key in a CIM Instance (C_i) contribute to an update.

2.2 Types of Tracking

CIMDIFF accepts two CIM repositories as input and tracks the differences. A CIM repository is accessible through standard CIM Client using credentials (URL, User, Password, Interop Namespace). Difference tracking provided by CIMDIFF are across two dimensions. First dimension provides two categories of difference tracking: i)Syntactic ii) Semantic. Second dimension provides two categories: i)Spatial ii)Temporal.

Syntactic: CIMDIFF tracks the syntactic difference between two CIM Repositories by exploiting the syntax of CIM structure. This mode is helpful for system management tool developers and testers to perform automated testing.

Sematic: CIMDIFF tracks the semantic difference between two CIM Repositories by using the combination of CIM structure and domain knowledge. This mode is helpful for system administrators.

Spatial: CIMDIFF tracks the spatial difference between two CIM Repositories that belong to two devices that are of same or different model. For example, spatial difference can be tracked between two IBM DS8000 storage subsystems. Spatial difference can also be tracked between a IBM DS8000 storage subsystem and a EMC Symmetrix storage subsystem.

Temporal: CIMDIFF tracks the temporal difference between two CIM Repositories of the same device that are captured at different point-in-time. For

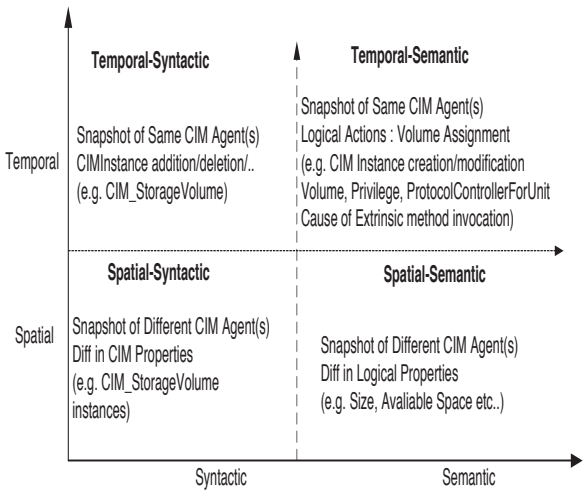


Figure 2: CIMDIFF Dimensions

example, temporal difference for a IBM DS8000 storage subsystem can be tracked between its current state and its state before one week.

With the combination of above two dimensions, CIMDIFF can track difference between two CIM repositories in four different fashion as described in Figure 2.

In the context of this paper, we use a tool called iSAN [26]. We have developed iSAN and open-sourced it through Eclipse Aperi [12] that allows to capture point-in-time snapshot of the complete CIM Agent i.e. the CIM Repository and host it. CIM Repositories created at different point in time capture device configuration changes due to regular data center tasks such as capacity provisioning, performance tuning, volume migration etc..

3 System Overview

This section provides an overview of CIMDIFF. It discusses architecture, inputs, outputs and the key components of the tool in the following subsections. CIMDIFF can reside on a separate server or can be integrated into the CIM agent to provide an integrated time-travel feature of configuration changes similar to the idea of ITIL's [24] (Information Technology Infrastructure Library) CMDB (Configuration Management Database) at the device level. CIMDIFF relies heavily on open standards and vendor extensions of open standards to derive the differences.

3.1 Architecture

Using Eclipse Aperi SAN Simulator (iSAN snapshotting framework) [26, 12], CIM repository can be persisted in a flat file or in a relational database. Device CIM agent is accessible via standard CIM Client [15] using *URL, User, Password, Interop Namespace*. *URL* is composed of *Protocol, IP address* and *Port*. CIMDIFF is a browser based web application that requires two CIM agent credentials as input to compute the difference. CLI interface of the tool is also exposed to the user. We use the terminology *Source CIM Agent* and *Target CIM Agent*. If both source and target CIM agent are pointing to different devices, CIMDIFF computes the spatial difference. CIMDIFF computes temporal difference, if both source and target CIM agents are pointing to the exact same device. Each device has a live CIM agent that reports the current information. Earlier point-in-time snapshots are captured and hosted by iSAN.

For example: an external `CIM Agent` deployed on a server reports two `IBM DS8000 Storage Subsystems`. Snapshot of this `CIM Agent` will be represented as one `CIM Repository` . `CIM Repository` will be associated with several `CIM Classes` such as `IBMTSDS-ComputerSystem` and `IBMTSDS-Volume` etc.. For each of these `CIM Classes`, there will be one or more `CIM Instance`. Each `CIM Instance` will have one `CIM Objectpath`. On a parallel hierarchy, there will be two `CIM Devices / CIM-ManagedElement` associated with the `CIM Repository`. Each `CIM Device` will have associated `CIM Classes`. Each `CIM Class` will have one or more `CIM Instances` associated with respect to the `CIM Device`. These two hierarchies essentially are created to answer CIM queries:

```
enumerate(CIM-StorageVolume)
associate(CIM-ComputerSystem ->
CIM-StorageVolume)
```

Basically, `CIM Repository` is a XML document with multiple hierarchy (Nodes having multiple parent) denoted by solid and dotted arrows in Figure 3.

There has been a lot of work on difference tracking algorithms for text data [27, 28], for relational data [29], for tree or XML data [30, 31]. Our tool is built on the base of text and XML difference tracking but different because of the domain specific meaningful change tracking functionality rather than the text or XML difference tracking. In addition, multi-parent nature of the nodes in the XML structure and the semantic domain knowledge of `CIM Recipes` that derive the difference makes our tool unique. Figure 4 describes the functional blocks of this tool. CIMDIFF has four main functional blocks: i)Hash Maker ii) KnowledgeBase iii) Hierarchy Resolver iv) Difference Tracker.

Hash Maker component calculates the hash using the popular SHA-256 or MD5 algorithm for the `CIM Repository`. Hash values are created and stored

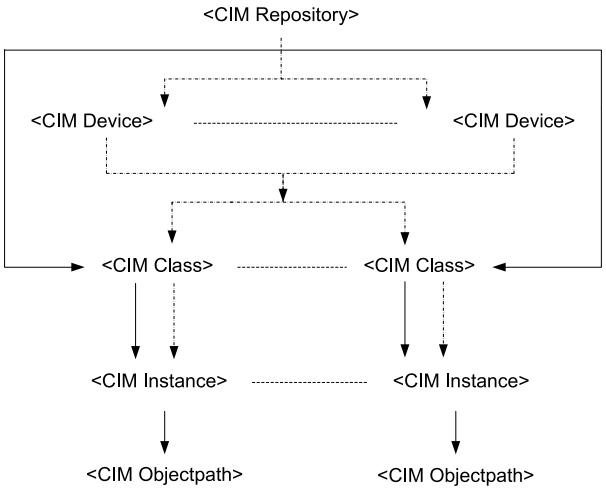


Figure 3: CIM Repository Structure

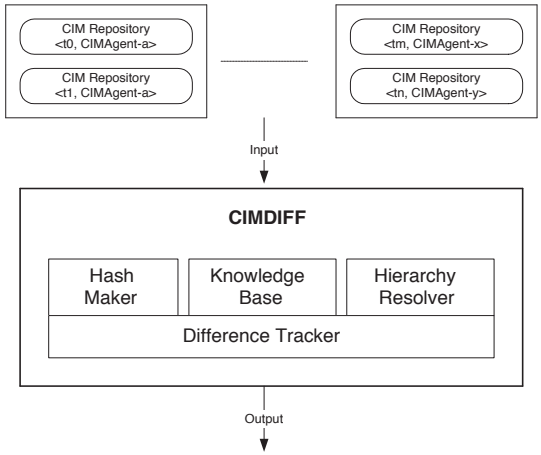


Figure 4: CIMDIFF Components

respective to nodes. Each node can contain multiple hash values since it can be parent of multiple nodes (CIM Agent Hash, CIM Device Hash). Hash values are built using bottom-up approach. Hash value of a parent node N is calculated from the hash values of the children nodes n. Leaf nodes that contain the hash values are the `CIM Objectpath`. Hash value calculation can also be integrated with the snapshot making process. Otherwise, it can be created later for a `CIM Repository` and correlated against it. Hash Maker component makes the difference tracking process efficient because, the hash values stored at nodes help CIMDIFF determine the tree/subtree isomorphism much efficiently. Tree/subtree isomorphism determination helps in the temporal-syntactic difference tracking. This component helps CIMDIFF traverse the least number of nodes necessary to perform difference tracking.

KnowledgeBase component stores the standard recipe invocation sequences and the changes associated with it. For example, if a StorageVolume was created and assigned (masked/mapped) to a server by execution of `CIM Recipe(s)` meaning a set of extrinsic methods were executed. Extrinsic method invocation on a `CIM Agent` results in `CIM Instance` creation/deletion/modification. Goal of this component is to store the popular standard recipe formats and their effect on `CIM Instances`. This component helps track the temporal-semantic changes across `CIM Repositories` of same `CIM agent(s)`. Similarly, any vendor-extended properties are also canned into KnowledgeBase for spatial-semantic difference tracking. KnowledgeBase is populated with standard SMI-S recipes to start with. It is updated with the latest variance in recipe formats as well as any new recipes that are introduced. KnowledgeBase can be centrally created and then distributed across installations to reflect the updates. Users can also edit custom recipe formats into knowledge fragments and update their KnowledgeBase. An example SMI-S recipe for storage volume creation is described in Figure 5

HierarchyResolver component is a wrapper around a MOF [11] parser. It traverses the CIM hierarchy through the `CIM Class` hierarchy and filters out the string mismatches. This component helps track the spatial-semantic changes across `CIM Repositories` of different `CIM Agent(s)` of similar type. For example: two storage pool instances (one from IBM DS8000 and other from IBM DS4000) have different vendor-extension `CIM Class`; `IBMTSDS-VolumeSpace` and `LSISSI-StoragePool` respectively. This module helps discard the regular string differences and picks the relevant difference in properties such as `Total Space` , `Total Available Space`. This module performs this task by i) resolving hierarchy since both these classes belong to same super class `CIM-StoragePool` ii) checking the KnowledgeBase for relevant `CIM Properties`

Difference Tracker component works in conjunction with the three components described earlier. It orchestrates the invocation of the CIMDIFF components. This component implements the algorithm to compute the minimum-cost optimal edit script. Outline of CIMDIFF algorithm is explained in Figure 6.

Edit script is then grouped into clusters based on the rules defined in KnowledgeBase to derive meaningful changes from the device configuration perspective. `CIM Objectpath` is used as the main correlation mechanism in the aggregation process to avoid ambiguity. CIM Properties are interpreted based on the rules to derive semantic differences. It also presents the output to the user and provides primitives for analysis across dimensions described in Figure 2 via both browser and java swing based graphical user interface. This component in con-

```
//Create Storage Volume on an Array [10GB RAID-5 Volume]
// RequestedSize = 10 * 1024 * 1024 * 1024 // 10 GB

1. Enumerate all StoragePools associated with $StorageArray
2. For each StoragePool {
     if(StoragePool is non-primordial) {
       get StorageCapabilities associated with StoragePool
       if(StorageCapabilities == Volume Requirements) {
         if (StoragePool has free space available) {
           invoke GetSupportedSizes and(or) getSupportedSizeRange
           //Round up using integer arthimetic
           get valid capacity of newly created volume
           Select StoragePool and Select VolumeSize }
       } } }
3. Services[] = AssociatorNames(StorageArray)
4. StorageConfigurationService = Services[0]
5. InArguments[ElementType] = 2 // Storage Volume
   InArguments[Goal] = GeneratedStorageSetting
   InArguments[Size] = VolumeSize
   InArguments[InPool] = StoragePool
   InArguments[TheElement] = null
   ReturnValue = InvokeMethod(
                   StorageConfigurationService,
                   CreateOrModifyElementFromStoragePool,
                   InArguments, OutArguments)
6. if ((ReturnValue == 0 || ReturnValue == 4096 &&
   OutArguments[TheElement] != null)) {
       //Reference to the newly created volume
       CreatedVolume-> = OutArguments[TheElement] }
```

Figure 5: Recipe - Storage Volume Creation

junction with HierarchyResolver also tracks the change of CIM model definition [11].

Figure 7 explains the input specification of CIM agent information to the tool. Spatial difference can be tracked between two similar types of `CIM Repositories` but not necessarily of same model. Two input CIM agent can report two storage subsystems or two servers. Spatial difference could be either syntactic or semantic. Spatial difference tracking is helpful for checking the difference in characteristics and configuration across similar devices. Spatial-syntactic difference tracking would derive information like i) the difference in number of `CIMInstances` per class between two `IBM DS4000` storage subsystems or between two different servers or between a `IBM DS4000` and a `IBM DS6000` storage subsystem ii) difference in `CIM Properties` iii) difference in `CIM Property` value. Same information in spatial-semantic form would be presented as i) difference in number of storage volumes, storage pools, file systems ii) difference in total available space, difference in total consumable space, difference in server main memory (RAM).

Temporal difference can be tracked between `CIM Repositories` of same `CIM Agents` snapshotted at different point in time. This kind of information is helpful for tracking the temporal change in the characteristics and configuration of the same device. As shown in Figure 7, point-in-time snapshots of CIM agent can be captured and hosted using the open source tool iSAN. Temporal-syntactic difference would present information such as i) the difference in number of `CIMInstances` per class ii) dif-

```
calculateMinCostEditScripts(C^r1,C^r2)
{
  /* Hashing */
  Map C^r1 to T^1, Map C^r2 to T^2
  Hash T^1, Hash T^2
  /* Traverse CIM Agent Hierarchy */
  /* Traverse Managed Device Hierarchy */
  Traverse CIM Agent hierarchy &
  Traverse Managed Device hierarchy
  {
    /* Check tree isomorphism based on Hash */
    /* Compute Insert, Delete, Update */
    for all Node N in T^1 and T^2
      for all CIMClass(es)
        for all CIMInstance(s)
          Compare CIMObjectpath
            Compare CIMProperties
  }
}
```

Figure 6: Outline of minimum-cost optimal edit script computation

ference in `CIM Properties` because of configuration change of the device such as volume creation, masking/mapping for storage subsystem or file system creation on server. Large amount of change in number of `CIM Instances` and/or `CIM Properties` due to device configuration change and its presentation via temporal-syntactic change tracking might be useful for systems management solution test automation but can be overwhelming for a system administrator. Grouping these changes by logical actions such as storage provisioning, file system provisioning is termed as temporal-semantic change.

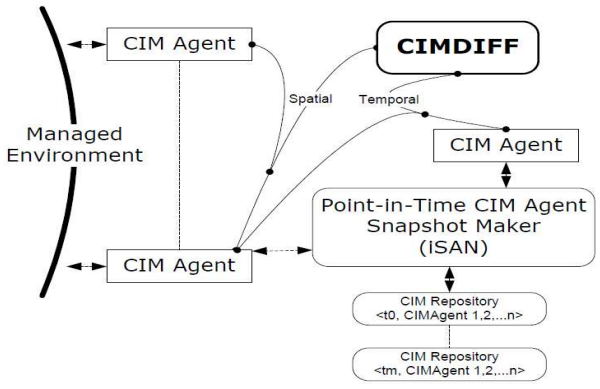


Figure 7: Input for CIMDIFF

4 Experimental Evaluation

Our experimental test bed is part of a production SAN environment. It contains servers with Linux and Windows operating systems. Production SAN also contains interconnecting fiber channel switches from multiple vendors, IBM and non-IBM storage subsystems. Each server has its own `CIM Agent` hosted on the server itself. Most of the fiber channel switches have external `CIM Agents` reporting the fabric information except a very few switches that has embedded `CIM Agent`. IBM and non-IBM storage subsystems have external `CIM Agents` hosted on separate servers. Management information is retrieved from `CIM Agents` using standard SBLIM CIM Client [15]. In this managed environment, server `CIM Agent` have a 1:1 mapping of `CIM Agent` to managed element. For fiber channel switches and storage subsystems, this environment has a maximum of 1:4 mapping of `CIM Agent` to managed device. CIMDIFF uses iSAN [26] to create the snapshots of the `CIM Agents`. `CIM Repositories` created from the snapshot process were stored in IBM DB2 UDB relational database. CIMDIFF also uses embedded database Derby [25] for handling data centers with very few devices. We integrated the HashMaker module of CIMDIFF with iSAN [26] to create the hash at relevant nodes during the snapshot process itself. Calculating hash did not impose a major overhead in terms of snapshot creation time. Calculating hash based on the standard CIM hierarchy denoted by solid arrows in Figure 3 introduced very minimal overhead during the snapshot process. Creation of device based hash denoted by dotted arrows took most of the time. An evaluation of hash creation overhead during a snapshot creation process for a `CIM Repository` of 2887 `CIM Instances` is described in Figure 8.

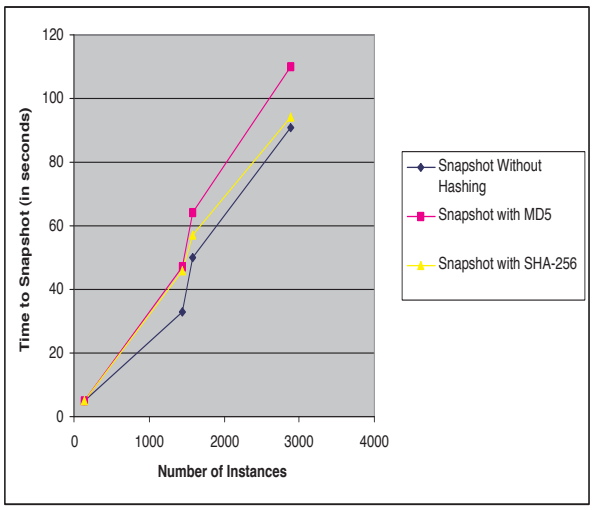


Figure 8: Hash Creation Overhead

CIMDIFF is implemented completely in java. It is hosted as a servlet based web application in a IBM Websphere Application Server container. Browser based user interface and user options are shown in Figure 9. CIMDIFF provides user interface and primitives to manage the CIM agent credentials. It also provides interface to manage and host snapshots of CIM agents. Knowl-

Table 1: Setup (Spatial Difference)

CIMDIFF	
CIM Repository	Description
CIMRepository-An	CIMAgent-A Storage Subsystem [IBM DS6000]
CIMRepository-Bm	CIMAgent-B Storage Subsystem [IBM DS4000]

edgeBase can be easily updated by uploading knowledge fragments. Knowledge fragments can be written by following a very simple XML based rule engine.

Based on the input CIM agent credentials, CIMDIFF automatically detects whether the mode is spatial or temporal. As described in Figure 9, syntactic and semantic output are presented in tabbed format. Due to the large and detailed nature of the output returned by CIMDIFF, we present selective portion of it in a tabular fashion to depict the nature of difference tracking.

Spatial Difference Tracking: Source CIM agent and target CIM agent credentials are supplied by the user (as shown in Figure 9). Since, both the CIM agents point to two different IBM DS8000 storage subsystems, spatial difference option is highlighted. Spatial difference can be evaluated between two similar types of devices, e.g. between a IBM DS4000 and a IBM DS8000 storage subsystem. Table 1 shows the setup for spatial difference tracking and Table 2 shows the types of syntactic differences that are tracked. Automation test suites used by discovery engines of system management tools [20] use the syntactic results to verify the proper operation of the tools.

Semantic information is derived based on the standards and(or) the KnowledgeBase. Standard CIM class CIM-StoragePool has CIM Properties `TotalManagedSpace` and `RemainingManagedSpace`. By accumulating the values across all the storage pools, TotalSpace and FreeSpace are calculated. Similarly, by using standard properties from `CIM-StorageSetting`, RAID level is calculated. These properties are common across vendor implementations(IBM storage subsystem represents its storage pool through the class IBMTSDS-ExtentPool or LSISSI-StoragePool based on the model. But, both these classes extend from CIM-StoragePool). CIMDIFF refers to rules across vendor extended properties to derive meaningful semantic differences. Rules can be easily added and updated by specifying new rules in simple XML format. Sample semantic difference values are shown in Table 3

Temporal Difference Tracking: If both source and tar-

Table 2: Spatial Syntactic Difference

CIMDIFF	
CIMRepository-An	CIMRepository-Bm
74 CIM Classes	78 CIM Classes
1874 CIM Instances	2436 CIM Instances
IBMTSDS-Volume PowerOnHours CIM Property	LSISSI-StorageVolume NO Corresponding Property
IBMTSDS-Volume NO Corresponding Property	LSISSI-StorageVolume RaidLevel CIM Property
IBMTSDS-Volume 212 CIM Instances	LSISSI-StorageVolume 120 CIM Instances
IBMTSDS-ExtentPool 7 CIM Instances (Extends from: CIM-StoragePool)	LSISSI-StoragePool 5 CIM Instances (Extends from: CIM-StoragePool)
IBMTSDS-DiskDrive 36 CIM Instances	LSISSI-DiskDrive 12 CIM Instances

Table 3: Spatial Semantic Difference

CIMDIFF	
CIMRepository-An	CIMRepository-Bm
Remaining Managed Space 840GB	Remaining Managed Space 300GB
Remaining RAID-5 Space 640GB	Remaining RAID-5 Space 300GB
Remaining RAID-1 Space 140GB	Remaining RAID-1 Space 0GB

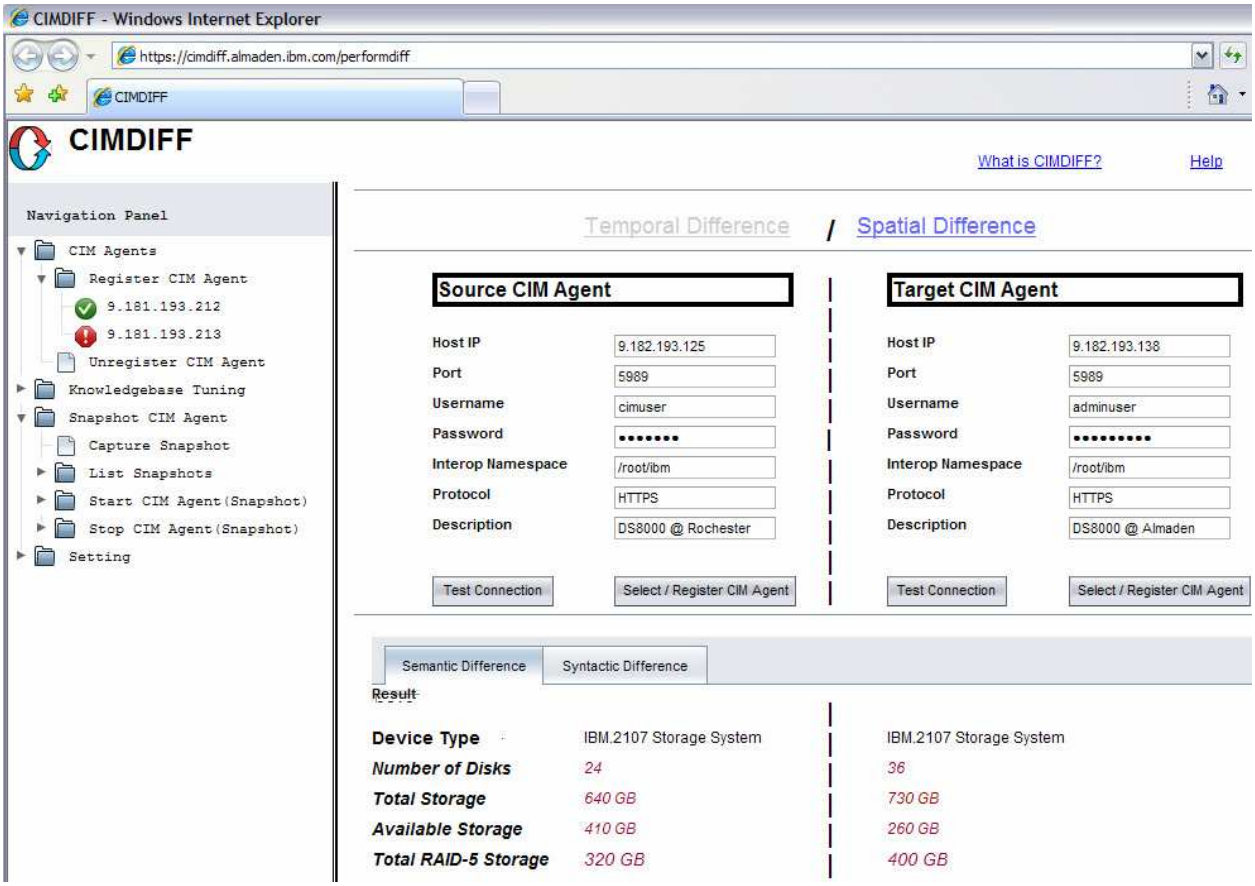


Figure 9: CIMDIFF User Interface

Table 4: Setup (Temporal Difference)

CIMDIFF	
CIM Repository	Description
`CIMRepository-A0`	`CIMAgent-A` Storage Subsystem [IBM DS6000] Snapshot at time `t0`
`CIMRepository-A1`	`CIMAgent-A` Storage Subsystem [IBM DS6000] Snapshot at time `t1`

get CIM agent point to the same device, CIMDIFF automatically detects and derives the temporal difference. Table 4 depicts the setup for a temporal difference scenario.

Due to the configuration actions performed on the storage subsystems such as volume creation/deletion, volume assignment/unassignment or zoning actions on fiber channel switches, temporal differences are reflected in the `CIM Repositories`. Table 5 shows some of

Table 5: Temporal Syntactic Difference

CIMDIFF	
`CIMRepository-A0`	`CIMRepository-A1`
1874 `CIM Instances`	1942 `CIM Instances`
`IBMTSDS-Volume` 212 `CIM Instances`	`IBMTSDS-Volume` 224 `CIM Instances` [SAME] 216 `CIM Instances` [DELETED] 5 `CIM Instances` [NEW] 17 `CIM Instances` [MODIFIED] 1 `CIM Instances`

Table 6: Temporal Semantic Difference

CIMDIFF	
CIMRepository-A0	CIMRepository-A1
1874 CIM Instances	1942 CIM Instances
	VOLUME CREATION VOLUME ASSIGNMENT
IBMTSDS-Volume 212 CIM Instances	IBMTSDS-Volume 224 CIM Instances
IBMTSDS-Privilege 18 CIM Instances	IBMTSDS-Privilege 20 CIM Instances
IBMTSDS- ProtocolControl lerForUnit 48 CIM Instances	IBMTSDS- ProtocolControl lerForUnit 52 CIM Instances
....	....

the syntactic difference due to the configuration actions. These difference are useful for automated testing validation of management software.

Modification of CIM Instance is derived because CIMObjectpath did not change but one or more of the other non-key CIMProperties changed across time.

In this scenario, syntactic difference might be overwhelming and does not provide with much meaningful information for administrator. But, by tracking the change in CIMInstances and then grouping the changed instances based on the recipe KnowledgeBase, we can show a potential list of extrinsic methods that were executed. As shown in Table 6, change between time t0 and time t1 was because of volume creation, volume assignment(masking, mapping). Correlating this information through the CIMObjectpath provides more value in terms of configuration change activities on the device. Sample KnowledgeBase is described in Table **??**

We used this tool to capture snapshot of our data center (small scale SAN environment) to i) track the temporal configuration changes for devices ii) compare and contrast similar type device configuration through spatial changes. This provided the system administrators with deep insight about change in the data center environment. Systems Management solutions [21] or Storage Resource Management solutions [20, 13] also provide this kind of feature at a much higher and better granularity because these solutions correlate data from across CIM Agents. CIMDIFF does not correlate data from across CIM Agents (e.g. addition/ removal of a port-to-port connectivity tracking by correlating the server port and fiber channel port). CIM Agent Snapshots of multiple CIM Agents can be aggregated together for evaluation. This distinction of aggregation versus correlation is a known limitation of CIMDIFF. We have also tested CIMDIFF and did not encounter any scalability issues for few hundreds of thousands of CIM Instances in the CIM Repositories.

5 Discussion and Related Work

General problem of detecting changes from snapshots of textual, relational or hierarchical structured (XML) data has been studied in great details. GNU diff utility is a popular tool in this context. Our calculation of edit scripts with respect to CIM Classes based on CIMInstance and CIMObjectpath is similar to the notion of LCS (Longest Common Subsequence) used in the difference tracking of CVS. Meaningful change detection algorithms [30, 31] have also a similar notion of difference tracking. Comparison of CIM standards and the vendor reported CIM Instances have a close analogy with XML schema / DTD and well-formed XML documents. In [33, 34, 37], change detection problems have been addressed for ordered trees. Authors [35] have also proved the meaningful change detection complexity to follow quasi linear time for NP-hard scenarios. Similarly, algorithm [30] also provides a similar approach by transforming the change detection problem to a problem of computing a minimum-cost edge cover of a bipartite graph. CIMDIFF is effectively domain specific logical difference tracking that is guided by industrial standards and customizable knowledge base as compared to the traditional text difference tracking, XML document difference tracking or change detection. Similar to our HashMaker approach, there has been similar work in the XML document change detection area that deals with constructing node signatures using exclusive-or (XOR) [32].

In virtualized environments [36], configuration change and movement of virtual resources are more often than a physical environment. CIMDIFF can very easily [39] show the difference in change in environment due to migration, provisioning [36] etc..

CIMDIFF provides an important primitive to the data center administrator as well as a very vital test tool for System Management solution developers and testers. SNIA CTP [23] can also potentially use this feature to make the testing process more efficient. CIM Agents before and after incorporating the changes to CIM Provider during testing iteration conformance cycle can use this tool. CIMDIFF provides valuable difference tracking powered by the knowledgebase. Currently, knowledgebase is manually populated by encoding the information from standard CIM recipes. Execution patterns of standard CIM recipe and their effects can be automatically derived by using machine learning. We

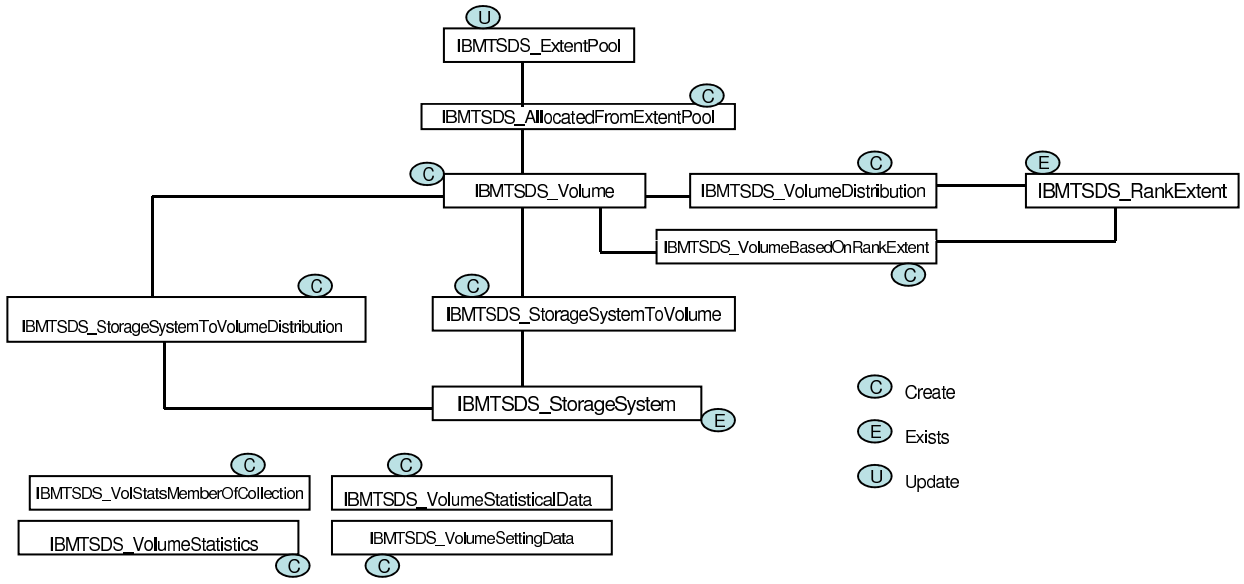


Figure 10: KnowledgeBase - Volume Creation

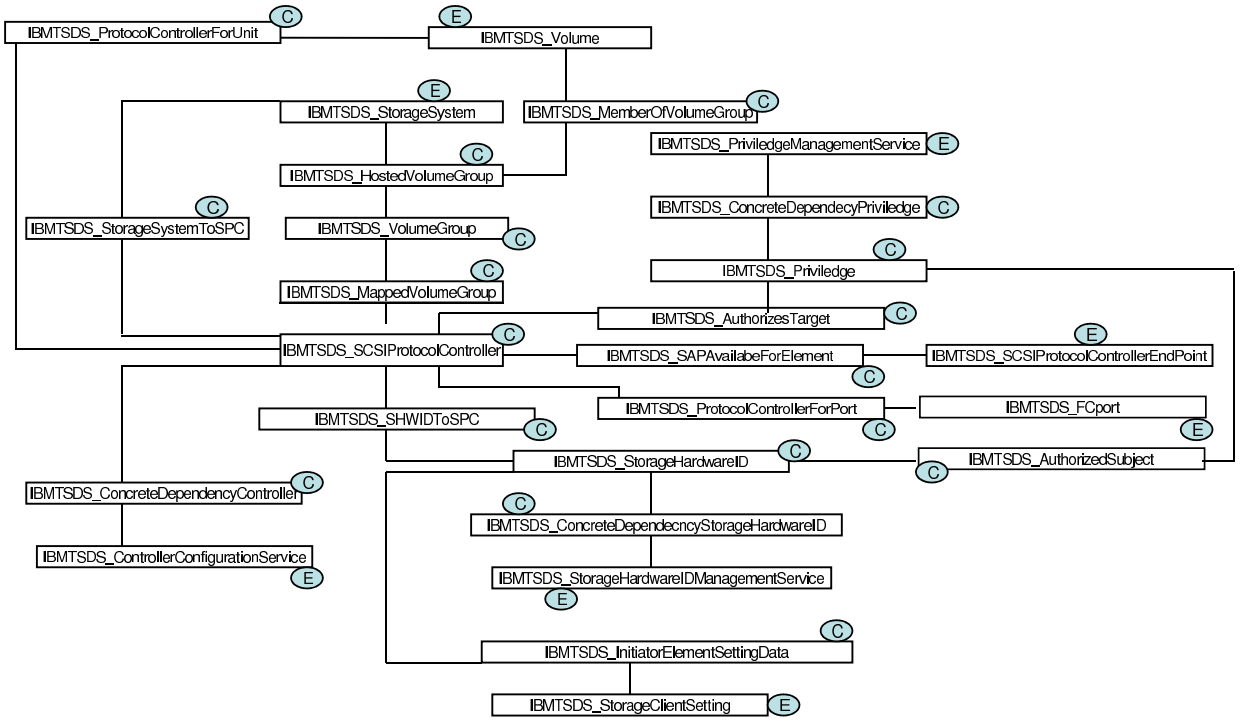


Figure 11: KnowledgeBase - Volume Assignment

are also extending a semi-autonomic interface for this tool to let the administrator create a knowledge fragment by grouping a discrete set of syntactic difference and associating with a particular semantic configuration change from the output panel of this tool itself.

Without creating snapshot of `CIM Agent` at intervals, change information also can be derived by: i) if `CIM Agent` supports lifecycle indication monitoring ii) by continuous subscription to all indications. But, such mechanism could be costly and would not provide semantic difference.

6 Conclusion

In this paper we presented an architecture, working of CIMDIFF that provides an important tool to system administrators with tracking data center configuration and characteristic changes. CIMDIFF also demonstrates the difference tracking based on the domain knowledge of CIM [3] standard. It builds on the standard text difference and XML difference tracking technologies. CIMDIFF uses an efficient hashing and knowledgebase interpretation on top of the multi-hierarchical XML data. Demonstrated approach provides an interesting solution to the nested structure that poses NP-hard problem.

CIMDIFF also limits itself to `CIM Repositories` aggregation rather than correlation functionality provided by systems management solution. CIMDIFF definitely provides an important technology for management solution testing as well as conformance [23] testing. Since most of the modern day data center entities have CIM agents, CIMDIFF offers a neat and quick tool for administrators in the data center.

References

[1] Amazon Elastic Compute Cloud EC2. `http://aws.amazon.com/ec2/`

[2] Amazon Simple Storage Service S3. `http://aws.amazon.com/s3/`

[3] Common Information Model (CIM). `http://www.dmtf.org/standards/cim`

[4] Distributed Management Task Force (DMTF). `http://www.dmtf.org`

[5] Storage Management Initiative Specification (SMI-S) `http://www.snia.org/forums/smi/tech_programs/smis_home/`

[6] Storage Networking Industry Association (SNIA). `http://www.snia.org`

[7] Systems Management Architecture for Server Hardware (SMASH) `http://www.dmtf.org/initiatives/smash_initiative/`

[8] IETF Simple Network Management Protocol (SNMP) `http://www.ietf.org/rfc/rfc1157.txt`

[9] Web-Based Enterprise Management (WBEM). `http://www.dmtf.org/standards/wbem/`

[10] Windows Management Instrumentation (WMI) `http://msdn.microsoft.com/en-us/library/aa384642(VS.85).aspx`

[11] Managed Object Format (MOF). `http://www.dmtf.org/education/mof/`

[12] Eclipse Aperi Project. `http://www.eclipse.org/aperi`

[13] EMC Control Center. `http://www.emc.com/products/family/controlcenter-family.htm`

[14] SNIA Interoperability Lab.. `http://www.snia.org/forums/smi/tech_programs/lab_program/`

[15] SBLIM CIM Client. `http://sblim.wiki.sourceforge.net/CimClient`

[16] Pegasus CIMOM. `http://www.openpegasus.org/`

[17] Sun WBEM. `http://wbemservices.sourceforge.net/`

[18] SNIA CIMOM. `http://www.opengroup.org/snia-cimom/`

[19] Hewlett Packard Systems Insight Manager (HP SIM). `http://h18002.www1.hp.com/products/servers/management/hpsim/index.html`.

[20] IBM TotalStorage Productivity Center (IBM TPC) `http://www-306.ibm.com/software/tivoli/products/totalstorage-data/`

[21] IBM Systems Director `http://www-03.ibm.com/systems/management/director/`

[22] Microsoft System Center `http://www.microsoft.com/systemcenter/en/us/default.aspx`

[23] SNIA Conformance Testing Program `http://www.snia.org/forums/smi/tech_programs/ctp/`

[24] ITIL `http://www.itil-officialsite.com/home/home.asp`

[25] Apache Derby `http://db.apache.org/derby/`

[26] R. Routray, S. Gopisetty, P. Galgali, A. Modi and S. Nadgowda. iSAN: Storage Area Network Management Modeling Simulation In *Proceedings of IEEE International Conference on Networking, Architecture, and Storage (NAS)*, 2007

[27] R. Wagner, M. Fischer. The String-to-String Correction Problem In *Journal of the ACM, Volume 21, Issue 1*, 1974

[28] E. Myers. An O(ND) Difference Algorithm and Its Variations In *Algorithmica*, 1986

[29] W. Labio, H. Garcia-Molina. Efficient Snapshot Differential Algorithms for Data Warehousing In *Proceedings of the 22th International Conference on Very Large Data Bases (VLDB)*, 1996

[30] S. Chawathe, H. Garcia-Molina. Meaningful Change Detection in Structured Data In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1997

[31] Y. Wang, D. DeWitt, J. Kai. X-Diff: an effective change detection algorithm for XML documents In *Proceedings of 19th International Conference on Data Engineering*, 2003

[32] L. Khan, L. Wang, Y. Rao. Change Detection of XML Documents Using Signatures In *Proceedings of Workshop on Real World RDF and Semantic Web Applications*, 2002

[33] K. Zhang, D. Shasha. Simple fast algorithms for the editing distance between trees and related problems In *SIAM Journal on Computing* , 1989

[34] D. Shasha, K. Zhang. Fast parallel algorithms for the unit cost editing distance between trees In *Proceedings of the first annual ACM symposium on Parallel algorithms and architectures*, 1989

[35] D. Shasha, K. Zhang. Detecting Changes in XML Documents In *Proceedings of International Conference on Data Engineering*, 2001

[36] M. Dehus, D. Grunwald. STORM: Simple Tool for Resource Management In *Proceedings of 22nd Large Installation System Administration Conference*, 2008

[37] B. Nguyen, S. Abiteboul, G. Cobena, and M. Preda Monitoring XML data on the Web In *Proceedings of ACM SIGMOD*, 2001

[38] K. Begnum, M. Disney, E. Frisch, and I. Mevg Decision Support for Virtual Machine Re-Provisioning in Production Environments In *Proceedings of 21st Large Installation System Administration Conference*, 2007

[39] VMware CIM APIs. `http://www.vmware.com/support/developer/cim-sdk/`

Transparent Mobile Storage Protection in Trusted Virtual Domains

Luigi Catuogno[1], *Hans Löhr*[1], *Mark Manulis*[2], *Ahmad-Reza Sadeghi*[1], *Marcel Winandy*[1]
[1] *Horst Görtz Institute for IT Security*
Ruhr-University Bochum, Germany
{luigi.catuogno, hans.loehr, ahmad.sadeghi, marcel.winandy}@trust.rub.de

[2] *Technische Universität Darmstadt,*
Center for Advanced Security Research Darmstadt (CASED)
Germany
mark@manulis.eu

Abstract

Mobile Storage Devices, such as USB flash drives, offer a flexible solution for the transport and exchange of data. Nevertheless, in order to prevent unauthorized access to sensitive data, many enterprises require strict security policies for the use of such devices with the effect of rendering their advantages rather unfruitful.

Trusted Virtual Domains (TVDs) provide a secure IT infrastructure offering a homogeneous and transparent enforcement of access control policies on data and network resources, however, the current model does not specifically deal with Mobile Storage Devices.

In this paper, we present an extension of the TVD architecture to incorporate the usage of Mobile Storage Devices. Our proposal addresses three major issues: coherent extension of TVD policy enforcement by introducing architectural components that feature identification and management of transitory devices; transparent mandatory encryption of sensitive data stored on mobile devices; and highly dynamic centralized key management service. In particular we address offline scenarios allowing users to access and modify data while being temporarily disconnected from the domain. We also present a prototype implementation based on the Turaya security kernel.

Keywords: security, mobile storage devices, USB storage, trusted virtual domains

1 Introduction

Trusted Virtual Domains (TVDs) [22, 9] are the forthcoming framework for the implementation of multi-domain/single-infrastructure computer networks like centralized data centers, where computational resources from different owners share the same physical infrastructure, or single organizational LANs that span over different offices, branches or functional areas.

Amongst the strengths of TVDs is the transparent enforcement of access control policies — platforms and users logically assigned to the same TVD can access distributed data storage, network services, and remote servers without executing any additional security protocols, while the resources belonging to different TVDs are strictly separated and, thus, remain inaccessible.

In this paper, we extend the security concept of TVDs to capture the use of Mobile Storage Devices (MSDs) such as portable hard drives and USB sticks, which offer additional flexibility for the transport of data across multiple working locations and devices (e.g., work stations, printers, cell phones, cameras, etc.). The non-triviality of this task results from the diverse security risks with regard to the data stored on MSDs. For example, MSDs can be easily lost or stolen, and consequently the confidentiality of data becomes an issue. Once left unattended by the user, MSDs can be manipulated with the goal to breach the integrity of the data or to disseminate corrupted data or malicious code once the device is re-connected to the enterprise platform. Many security solutions for MSDs adopted in practice rely on a mixture of different techniques. In fact, the choice of appropriate mechanisms is guided by trade-off between their costs and offered benefits [33, 4]. Recent surveys indicate that existing security policies vary across organizations from none to very restrictive ones disallowing MSDs at all [16, 17, 45].

The deployment of MSDs is a challenging task for the current TVD model. Indeed, TVD infrastructures that want to take the major advantages of versatility of mobile storage devices have to address two main objectives: On the one hand, they should be *efficient* enough to reduce the overhead of enforcing security policies; on the other hand, they have to be *secure* enough to reduce the efforts requested to users and consequently reducing the effects of human errors.

Our Contribution In this paper we present an enhanced secure management model for MSDs within the framework of TVDs. Moreover, we present the design and the implementation of a comprehensive solution to enable transparent user-friendly encryption of sensitive data within a TVD. In particular, we address the usage of mobile storage devices to transport data within a domain by pursuing a separation between data storing and centralized key management. This separation is necessary to achieve offline data access, e.g., to allow a platform that is temporarily disconnected from the domain to process the data while preserving the desired security properties.

Paper organization We describe the problem definition in more detail in Section 2, and briefly overview the existing TVD concept in Section 3. Section 4 introduces our solution of integrating MSD management in TVDs, whereas we discuss the details of our MSD access control in Section 5. We describe our prototype implementation (Sec. 6), evaluate the security of our approach (Sec. 7), and discuss related work (Sec. 8). Section 9 concludes our work.

2 Problem Description

TVDs (see Section 3 for background) introduce a homogeneous and transparent infrastructure that aims at the separation between multiple domains with different security and trust policies. Enterprises and other organizations often have to deal with data of more than one security level. As a consequence, they separate their workflows to meet the different security requirements of their domains, e.g., working with confidential (internal) and public documents at the same time. The application of a TVD infrastructure can help these organizations to transparently enforce their security policies.

The incorporation and usage of mobile storage devices in TVDs would increase the flexibility of users in their workflows, but poses a challenging task in the design of the overall security architecture. MSDs are regularly employed to store copies of documents that the user may take home or to another office, raw data to be processed elsewhere, or on-the-fly data backups. In particular, MSDs are frequently used *offline*, i.e., plugged to any platform while it is not connected to the domain network (e.g., a laptop on the airplane).

MSD deployment raises several concerns about data confidentiality and integrity. Adversaries could intercept (steal) devices and try to read private data or even to make unauthorized changes. While encryption and digital signatures can achieve confidentiality and integrity of data stored on MSDs, the average human user is likely to be unskilled to properly configure and use standard security solutions. This may increase the probability of human errors and result in ineffective data protection. Moreover, users may feel any security policy as a nuisance that introduces overhead in their tasks and, therefore, try to circumvent or ignore it.

One important issue is that MSDs are passive components, thus enforcement of security policies relies on the computer they are connected to. We may assume the policy is correctly enforced as long as the MSDs are used within the TVD boundaries. This assumption is in general no longer true if any MSD is used outside its domain, e.g., when is connected to an outsider computer.

Our aim is to extend the TVD model with the benefits of using MSDs, allowing the transparent binding of an MSD to a certain TVD so that only platforms of the same TVD can access the stored data. Deploying MSDs within the TVD requires some refinement to the model due to the following concerns:

- *Device identification.* An MSD can move from a workstation to another without any control by the TVD infrastructure. Hence, whenever an MSD is plugged in, the platform should be able to distinguish the device and the domain this device belongs to.
- *Dynamic Device Management.* Unlike weighty storage devices, MSDs may unpredictably appear and disappear within the domain, according to the users' needs. This requires the introduction of an MSD management infrastructure in order to handle, e.g., creation and distribution of encryption keys.

2.1 The Offline Scenario

As mentioned above, MSDs are also used offline (i.e., the policy-enforcing platform is not connected to the domain), which introduces additional security problems. Almost all duties related to policy enforcement (e.g., authentication, key distribution, etc.) rely on interactive protocols. But policy rules may change, platforms may join/leave the domain (and should no longer access data), (disclosed) encryption keys may be revoked (and new ones should be generated and distributed). Whenever a policy change occurs, these changes have to be promptly propagated to all platforms in order to prevent further disclosure or sensitive data.

Hence, allowing offline platforms to access domain data stored on MSDs needs to fulfill the following security requirements:

- *Delegation*. Each domain platform should be able to enforce a policy (this means online and offline). For instance, each platform should store locally an instance of the policy and any credentials needed to enforce the policy.

- *Delayed revocation*. The notification of revocation of any platform, compartment, or device to offline platforms is delayed to the time they will re-connect to the domain network. In the meantime, data processed by these platforms and transferred over the domain through a mobile device may be made partially (or totally) invalid because of revocation. In order to validate data on mobile storage devices, every platform should be able to verify whether the data has been processed by a revoked platform.

- *Authentication and data integrity*. Access and data modification should be infeasible for outsiders.

- *Traceability and recovery*. Domain members should be able to track unauthorized data modifications and to reconstruct the previous data layout.

3 Background on Trusted Virtual Domains

Trusted Virtual Domains (TVDs) [22, 9] are a novel security framework for distributed multi-domain environments which leverages virtualization and trusted computing technologies. In this section we give a brief overview of the TVD concept and its features, and briefly introduce its main components and protocols.

In a virtualized environment, different applications and services together with their underlying operating systems are executed by different Virtual Machines (VMs) that share the same physical infrastructure. Each virtual machine runs in a logically isolated execution environment (which we call *compartment*), controlled by the underlying Virtual Machine Monitor (VMM). In such an environment, the user's work space is now executed by a virtual machine that is hosted by the VMM running on the physical platform along with other architectural components.

A TVD is a coalition of virtual machines that trust each other, share a common security policy and enforce it independently of the particular platform they are running on. Moreover, the TVD infrastructure contains the VMM and the physical components on which the virtual machines rely to enforce the policy. In particular, the main features of TVDs and the TVD infrastructure are:

- *Isolation of execution environments*. The underlying VMM provides containment boundaries to compartments from different TVDs, allowing the execution of several different TVDs on the same physical platform.

- *Trust relationships*. A TVD policy defines which platforms (including VMM) and which virtual machines are allowed to join the TVD. For example, platforms and their virtualization layers as well as individual virtual machines can be identified via integrity measurements taken during their start-up.

- *Transparent policy enforcement*. The Virtual Machine Monitor enforces the security policy independently of the compartments.

- *Secure communication channels*. Virtual machines belonging to the same TVD are connected through a virtual network that can span over different platforms and that is strictly isolated by the virtual networks of other TVDs.

Figure 1 shows an example of two TVDs that are distributed over different physical machines, and illustrates main components of the TVD architecture and their relations. The *TVD policy* is a set of rules that state security requirements a compartment should fit to be admitted to the TVD (e.g., integrity measurements of the platform and VMs) and defines both *intra*-TVD and *inter*-TVD information flow policy. A special node, namely the TVD Master, logically acting as a central server, controls the access to the TVD following the admission control rules stated in the TVD policy. The TVD Proxy is a compartment that locally enforces the TVD policy on the platform it is running on. Several TVDProxies, belonging to different TVDs can be instantiated on the same platform.

The process of TVD establishment in two steps, "deploy" and "join", is detailed in [29]: With the `TVD_deploy` protocol, the TVD Master verifies a platform and its ability to enforce the TVD policy. Then, in the `TVD_join` procedure, the TVD Proxy (verified by the TVD Master during the deploy phase) can verify virtual machines that are executed on the platform, and admit them to the TVD. Trusted computing technology is used to establish trust in the reported measurement values. For example – following the TCG approach – hash values of the software boot stack (including BIOS, bootloader, and virtualization layer as well as loaded virtual machines) are stored in and signed by a Trusted Platform Module (TPM) [43] and reported to the TVD Master during an attestation protocol. The TVD Master can reliably verify whether the reported values match the required ones of the TVD policy. Based on this, the TVD Master can implicitly rely on the enforcement mechanisms of the local platforms.[1]

Techniques to isolate and manage the virtual networks of different TVDs are given in [10]. Basically, virtual switches on each platform implement VLAN tagging for

[1] The definition of the required integrity measurement values in the TVD policy postulates knowledge about the behavior and security properties of the corresponding software programs. In practice, this can be achieved, e.g., through independent trusted third parties who evaluate and certify products according to evaluation standards like Common Criteria.

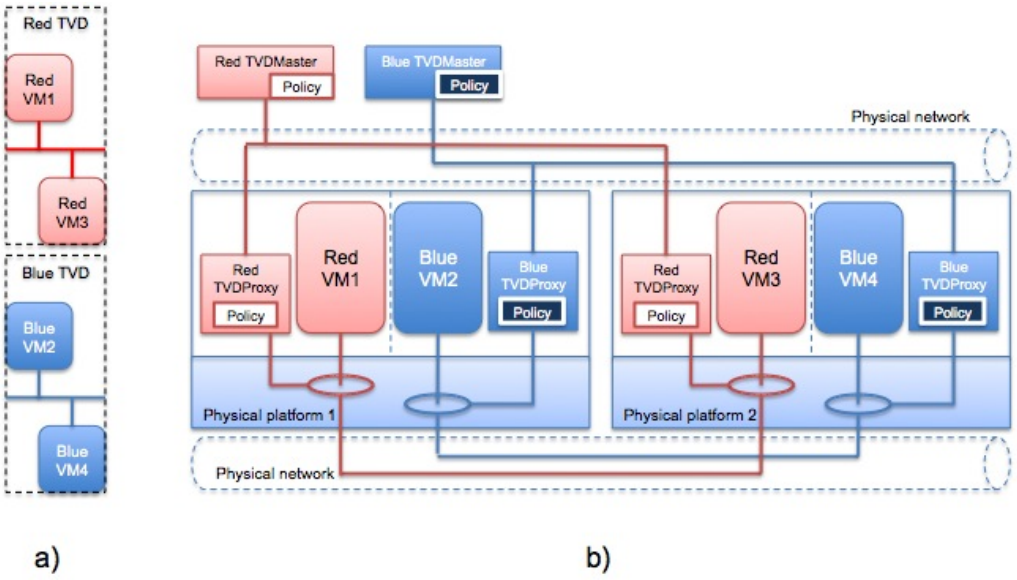


Figure 1: An overview of trusted virtual domains (TVDs)

Part a) shows the logical view of two TVDs, distributed over two physical platforms. Part b) shows the physical deployment of the TVD components, including the TVD Master .

local connections, and secure VPN for remote connections.

Various applications of TVDs were already shown and discussed in the literature. One example addresses the idea of applying the TVD concept for secure information sharing [26]. Other examples are virtual data centers [5], or enterprise rights management [20]. However, none of these works addresses the secure incorporation of mobile storage devices as we require.

3.1 Management of TVDs

The leading approach of management of TVDs within both centralized Virtual Data Centers and distributed organizational networks leverages on the deployment of advanced network management technologies (e.g., the *Web-based Enterprise Management* [14]) that provide highly integrated tools to accomplish administration tasks.

In a TVD-enabled infrastructure, management activities span over three levels. The *infrastructure level* concerns maintenance of physical resources, setup and configuration of the overall logical infrastructure, and assignment of resources to the different TVDs. At *domain level*, administrators take care of the TVD deployment, virtual machine setup and management of policies, devices and keys. Finally, at *compartment level*, running applications and current users can be notified of some events, coming from the underlying platform (e.g., revocation). At each level, administrators have an integrated management console that allows them to control all the operations under their responsibility. The administration of Virtual Data Centers with TVDs is discussed in [5].

4 Our Solution

In this section, we describe our solution to incorporate the use of mobile storage within the TVD framework. First, we describe how mobile storage operations are accomplished and, subsequently, we describe the new functionalities we introduce in the existing TVD architecture.

4.1 System Operation

Figure 2 shows an example TVD-enabled infrastructure in which two different TVDs are deployed. Each physical platform runs one or more virtual machines belonging to one of the existing TVDs. Several MSDs, variedly assigned to one of existing TVDs, are available to the users.

Here follows a typical usage example. The user Alice is working on the virtual machine VM_1 and plugs in her USB stick D_1 to the platform P_1 to make a backup copy of her files. Some specific components running on the platform P_1 (see Section 4.2.2) identify the plugged device, verify whether it has been assigned to the same TVD of VM_1 and retrieve the cryptographic keys that

are used to encrypt and decrypt data on it. If everything succeeds, the device is made available to VM_1.

At this point, a further refinement to the device access control can be achieved on a per-VM basis. To this end, a set of rules that defines access privileges to each device assigned to the TVD (*device access policy*), has been added to the TVD policy. For each device, these rules state which operations and privileges (e.g., read, write) are granted to each virtual machine in the same TVD.

Hence, the platform P_1 allows VM_1 to mount the device D_1 under the constraints stated by the device access policy (read-only, read-write). Finally, if it is consistent with access privileges of VM_1, the copy of Alice's data can take place.

We recall that both device identification and key retrieval are performed automatically and transparently by the platform when the device is plugged in. The guest operating system of VM_1 does not need any special software to open and access the device, and no additional operation from the user (e.g., further authentications besides login, or providing keys) is required to handle data contained on the device. Moreover, we stress that data encryption is mandatory, thus the user cannot choose to not encrypt data once the mobile storage device has been assigned to a TVD.

Data stored on D_1 can be accessed only by those virtual machines which joined the same TVD. In particular, let D_1 be plugged in to platform P_3 which runs two virtual machines, VM_3 and VM_4. The virtual machine VM_3, which is in the same TVD as D_1, can access D_1, whereas VM_4 cannot. Trying to access D_1 on a virtual machine from a different TVD leads to a failure, because the platform is not allowed to retrieve the corresponding encryption key.

4.2 Virtual Storage Management

The main idea of our approach is to add access rules and the management of cryptographic keys for mobile storage devices at those components which are already responsible for handling access rules and keys for the TVD network, i.e., adding the information to the TVD policy and performing the enforcement by the TVD Master and the TVD Proxies. Moreover, we add additional components to the virtualization layer of each platform to deal with the specifics of MSDs: the *MSD Manager* and a *vMSD* component. Hence, the trusted components of virtual storage management in a TVD are the TVD Master and, on each platform, TVD Proxy, MSD Manager, vMSD, and, of course, the virtual machine monitor. We explain the interaction of these components in the following subsections.

4.2.1 Device Identification

Information needed for the identification of an MSD is contained in a special data structure named *identification record*, and stored on the device along with the data provided by the user. This information includes the name of the TVD the device belongs to, and the *device-id*, which uniquely identifies the device within the TVD. The identification record is generated and stored on the device when it is initialized.

4.2.2 Device Key Retrieval

To each MSD, our architecture associates a *security record* containing some security related information (see Section 5.2), including encryption keys. Security records of all MSDs are indexed by the *device-id* and are stored in a special database: the *Domain Device Directory (DDD)*, placed at the TVD Master. On every platform, each TVD Proxy handles a *Local Device Directory (LDD)* that partially replicates the domain directory of its domain. Physical platforms run a stand-alone component: the *MSD Manager*, which waits for a device to be plugged in. When this happens, the MSD Manager reads the device's identification record and extracts the *device-id* and the name of the TVD it is assigned to. The MSD Manager checks whether the TVD Proxy for that domain is running on the platform, and if so, the MSD Manager requests it to fetch the security record for the plugged device from the LDD.

If the record is found, the TVD Proxy allows the MSD Manager to open the device and releases its keys. If the TVD Proxy cannot find the requested record, it forwards the request to the TVD Master, which in turn searches for the record in the Domain Device Directory and replies either the requested record or an error message. Finally, the TVD Proxy stores the received record in the Local Device Directory and goes on.

The Local Device Directory fulfills two important functions. The first one is: allowing offline platforms to open a subset of mobile storage devices assigned to their domain, provided the corresponding security records have been added previously. The second one is: avoiding that the TVD Master is queried every time an MSD is used within its domain.

4.2.3 Accessing Devices

As stated above, neither the user, nor the virtual machine (which runs a commodity operating system) need to perform any additional task to access the MSD. Indeed, in our architecture, plugging in an MSD to a platform looks like plugging in a plain mobile storage that stores data in clear, from the virtual machine's point of view.

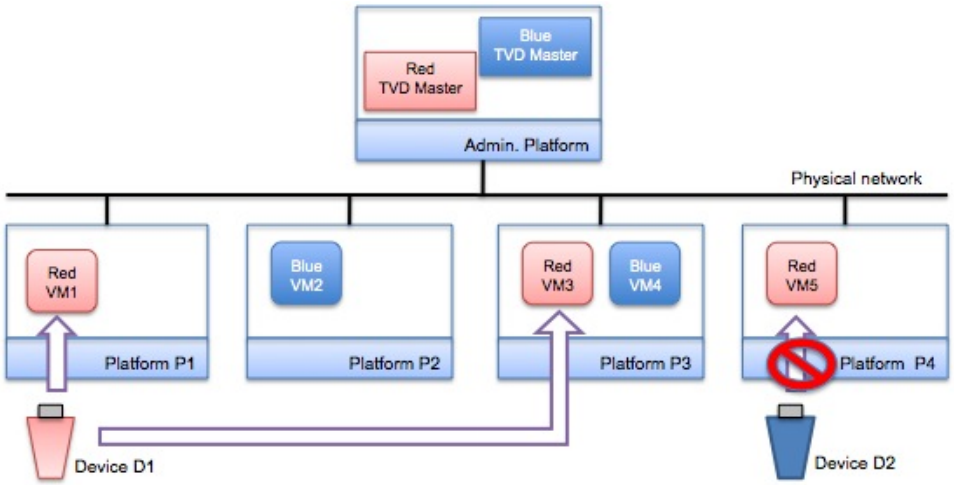


Figure 2: Example of using MSDs in an environment with two TVDs respectively named *red* and *blue*.

Data encryption (as well as device access policy enforcement) is performed by a specific component running on the platform: the *virtual MSD (vMSD)*. The vMSD features an encryption layer through which the VM mounts and accesses the device.

More precisely, a vMSD instance is created for each MSD plugged in to the platform and is given the corresponding keys by the TVD Proxy once the key retrieval has been completed successfully. Hence, the vMSD announces itself to the VM as a virtual device. All data the virtual machine reads/writes through the virtual device is silently processed by the vMSD layer and stored on the real MSD.

4.3 System Administration

4.3.1 Device Initialization

New mobile storage devices are assigned to a TVD through an initialization procedure. When an unassigned MSD is plugged in to a platform, the user is asked whether the system may initialize it.

The initialization requires the cooperation of the TVD Master. Indeed, the TVD Proxy running on the platform requires the TVD Master to generate the identification record and the security record (see Section 5.2) for the new MSD. The former is sent back to the platform and stored to the device via the vMSD whereas the latter is saved in the domain device directory and propagated to the requesting platform through the key retrieval procedure.

The device access policy can be determined at different levels. Users can explicitly provide the rules they need for their devices, or some general rules, stated both at platform or at domain level can be applied as default policy. Anyway, it is the TVD Master which writes the requested rules to the TVD policy.

When an MSD should be removed from a TVD, it can be de-initialized by simply deleting its security record from the Domain Device Directory (see Section 5.2).

4.3.2 Revocation

Any user, virtual machine or platform, may leave the TVD for administrative reasons or can be revoked because any kind of corruption has been discovered. In both cases, the administrator has to edit the TVD policy and any other involved data structures at the TVD Master (e.g. the Domain Device Directory).

Administrative revocations can be integrated within the setup and configuration procedures featured by the employed network management framework, so that, while modifying the layout of the network, administrators can consistently update the TVD policy.

The TVD architecture allows the TVD Master to realize whether platforms or virtual machines have been corrupted when they try to respectively deploy or join the TVD. The consequent failure can be notified to the administrator who can adopt the needed measures through the management facilities.

At the moment, the architecture presented in this paper does not feature any mechanism to automatically detect run-time intrusions. We discuss details of revocation in Section 5.5.

5 MSD Access Control Management

In this section we describe the realization of our MSD access control management. First, we briefly describe the enabling technologies, mainly cryptographic primitives

we use, followed by a description of the initialization phase, and how the access control of MSDs is handled. Last but not least we present the more advanced feature of key revocation.

5.1 Building Blocks

In our architecture we apply two cryptographic primitives: a symmetric encryption scheme with lazy revocation for data encryption and an identity-based signature scheme for data authentication. Our solution is intended to be independent from the employed cryptographic primitive, so we base our design on a general model like the one discussed in [2]. Therefore, we briefly recall terminology and notation needed in the following. For more details, we refer to [2].

5.1.1 Lazy Revocation

A group of users share some data encrypted with the same symmetric encryption algorithm. In general, a validity time (timeslot) is assigned to each key. So, if t is the current timeslot, all keys k_i generated at times $i < t$, are considered revoked. At time t, all group members know the *current* key k_t. Whenever a user leaves the group, the current key is revoked and the new key k_{t+1} is generated and delivered to the remaining group members. The lazy revocation concept is based on the assumption that protecting old data from revoked users is not necessary since they could have accessed the data already and disclosed it to outsiders or other parties. Hence, previously encrypted data are not re-encrypted, whereas new data will be encrypted with the new key in order to preserve confidentiality. Anyway, each user still needs old keys, to read data encrypted at previous timeslots.

To avoid that participants store all revoked keys, several schemes [3, 32] provide users with a single *user master key* K_t for each timeslot t. K_t can be used to *extract* all keys k_i $(0 \leq i \leq t)$. This kind of schemes is characterized by a *trusted status* for each timeslot t. The initialization algorithm of the lazy revocation scheme generates the initial engine state E_0 related to the timeslot $t = 0$. User master key K_0 is *derived* from E_0. When a revocation occurs, the scheme *updates* its state taking current state E_t to the new state E_{t+1}, hence, a new master key K_{t+1} is derived and delivered. Revoked users still know K_t, but cannot use it to *extract* the new key k_{t+1}.

5.1.2 Identity-Based Signature

Let $W = \{w_1, \ldots, w_n\}$ be a group of identities (of users or platforms), represented as binary strings. An Identity-Based Signature (IBS) scheme [23, 18] is initialized by a trusted Key Generation Center (KGC) which generates the *master secret key* SK and the corresponding *master public key* PK. Then, using SK and an identity w, KGC can derive the appropriate *secret signing key* SK_w, which it then securely transports to w. This allows w to generate own signatures σ_w on any message of its choice, which can be verified by others using the identity w and the master public key PK.

5.2 Initialization

For each TVD, there is a TVD Master, which is assumed to be always online in order to handle new key retrieval requests from the various platforms. The TVD Master creates and manages for each mobile storage device the states E_t and master keys K_t for lazy revocation, as well as the master secret key SK and master public key PK for the identity-based signature scheme. To allow each platform to verify signatures made by the TVD Master, we assume a public-key infrastructure that enables the TVD Master to issue certificates for new master public keys.

In particular, the initialization ("coloring") of a new mobile storage device $\mathcal{D}$ for a TVD works as follows. Let the TVD be identified by (have the color) *tvdID*. Assume $\mathcal{M}$ to be the TVD Master of *tvdID*. Once the blank device $\mathcal{D}$ is connected to a platform $\mathcal{P}$, the Virtual Storage Management of $\mathcal{P}$ formats the device and requests the local TVD Proxy belonging to *tvdID* to generate an identification record IR for the device. The TVD Proxy contacts the TVD Master to issue the record containing a newly generated device-id d and *tvdID*. Figure 3 shows the corresponding protocol.

Beside the creation of the identification record, $\mathcal{M}$ also initializes encryption and signature schemes for $\mathcal{D}$. $\mathcal{M}$ creates the tuple (E_0, PK, SK, W, RL), where E_0 is the initial state of the symmetric encryption scheme, PK is the master public key and SK the master secret key for the IBS scheme, W is the set of writers (it is given as input to the initialization procedure) and RL, initially empty, is the set of revoked writers. All these information associated to $\mathcal{D}$ are stored in a newly created entry in the *Domain Device Directory* (DDD) on $\mathcal{M}$.

$\mathcal{M}$ derives its own signing key SK_M from SK. $\mathcal{M}$ signs the identification record $(d, \textit{tvdID})$ under SK_M and sends the result $IR := (d, \textit{tvdID}, sig[SK_M](d, \textit{tvdID}))$ to the TVD Proxy, which in turn stores it to the device. Now we have $\mathcal{D}.id = d$ and $\mathcal{D}.owner = \textit{tvdID}$. The latter indicates to which TVD the mobile storage device is assigned, i.e., the "color" of the TVD. Note that storing files from different TVDs on the same device is logically equivalent to having one device for each TVD. Here, for simplicity, we consider only the second case.

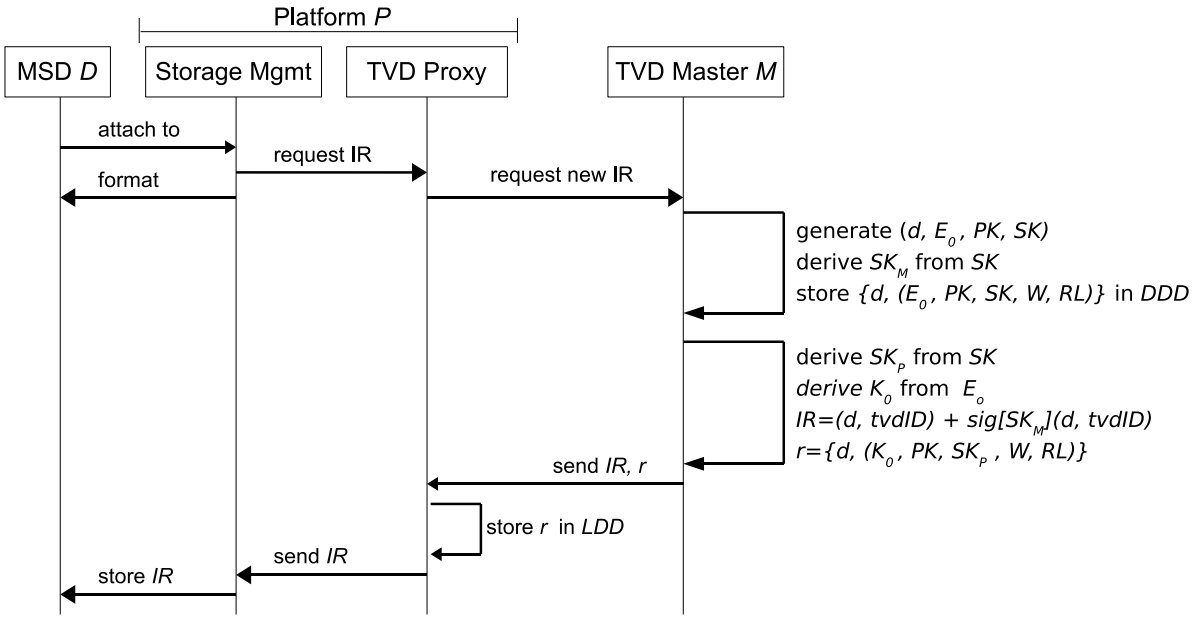


Figure 3: Device coloring protocol.

Note that neither E_i nor SK are delivered to any platform, they are stored and processed only on the TVD Master $\mathcal{M}$. Indeed, $\mathcal{M}$ distributes to each platform $\mathcal{P}$ the key management record $r = (d, (K_t, PK, SK_P, W, RL))$ where: K_t is the *current* master key for encryption, and SK_P is the signing key of the platform $\mathcal{P}$ which was derived from SK by the TVD Master. The record r is stored in the *Local Device Directory* (LDD) of the corresponding TVD Proxy on $\mathcal{P}$.

Device De-Initialization Finally, a device can be "uncolored" by deleting its identification record (by formatting it) and erasing its corresponding entries in the global (DDD) and local device directories (LDD). Entries in both directories can have an expiration time, to avoid that the TVD Master keeps information about devices forever.

5.3 MSD Access Control Mechanism

When a device $\mathcal{D}$ assigned to the TVD is attached to the platform $\mathcal{P}$, which hosts VMs of the same domain, then the Virtual Storage Management of $\mathcal{P}$ extracts the identification record IR from the device. If the device is recognized, i.e., $\mathcal{D}.owner$ is this TVD and the signature of IR is valid, then the MSD Manager requests the corresponding TVD Proxy to search for the record indexed by *d*=$\mathcal{D}$.*id* in its Local Device Directory in order to obtain the device keys. If the entry is not found because the device has not been attached to this platform yet before, the query is forwarded to the TVD Master $\mathcal{M}$.

5.4 File Storage

In a naive approach, a platform, once it has obtained required keys, gets the whole file from the device, decrypts it, verifies the attached signature and makes it available to user applications. Eventually, it encrypts and signs the updated file and copies it back to the device. This approach raises a problem: new data overwrite old data.

To fulfill our *traceability and recovery* requirement, we store data to the MSD using a *versioning filesystem*.

In a versioning filesystem, each file can exist in several versions. Usually, users can access transparently (unlike in conventional application-level revision control systems [41]) the latest file version as in a regular filesystem, whereas a set of user-level utilities feature several administrative tasks on file versions.

Versioning filesystems allow to record the history of changes to files in data repositories, and are useful where it is needed to maintain accurate logs of data flows and possibly to reverse some operations.

The versioning policy we adopted is known as *Copy-on-write*: a new file version is created each time it is modified, e.g., by a write operation. Hence, a node which accesses any input file f_i, saves (and signs) its version to the mobile storage device as a new file f_{i+1}, instead of simply overwriting the previous one. Afterwards, each node can load the latest version of any file for which it can successfully verify the signature.

As a consequence of this versioning policy users will progressively consume all the available storage space on any device. Therefore, the TVD policy also defines

a *purge* privilege that allows certain users to delete or merge old versions. The purge operation is rather critical, hence, it is intended to be done only by domain administrators and only when the device is connected to an online platform.

We embedded both data security (encryption and digital signatures) and handling file revisions into our architecture. The Virtual Storage Management performs the corresponding operations transparently with respect to user compartments.

5.5 Revocation of Cryptographic Keys

Both encryption and signing keys can be revoked in three cases:

- *Member revocation:* Whenever a platform, VM, or user is no longer member of the domain, the TVD Master updates the encryption key (and revokes the signing key if any).
- *Key disclosure:* Whenever it is known that a key has been disclosed to unauthorized parties (e.g., due to malicious users or compromised platforms), the corresponding key must be revoked.
- *Expiration:* Creating and updating keys are bound to a timer.

Suppose that at time t, revocation of k_t is requested, $\mathcal{M}$ updates the encryption engine taking it from state E_t to state E_{t+1}, derives the new master key K_{t+1}. K_{t+1} is delivered to platforms that can extract the new encryption key k_{t+1}.

To revoke the signing key SK_w, the TVD Master $\mathcal{M}$ adds w to the revocation list RL. If the revoked key has to be replaced by a new one, $\mathcal{M}$ generates a new writer-id w', puts it into the set W of write-enabled nodes and sends it to the node previously known as w. Moreover, $\mathcal{M}$ sends the new revocation list RL to all other platforms. All data signed with the revoked key SK_w are no longer accepted by any platform.

Key revocation may occur asynchronously with respect to device access and the periodical update requests by TVD members. Therefore we setup a key event notification system in which the TVD Master notifies a key revocation to all platforms hosting VMs of the domain by raising an appropriate event or alarm. Once the event notification has been received, each online platform renews its keys. Event notifications are queued, so that they can be delivered to offline platform once they connect to the TVD.

5.6 Offline Scenario

We briefly revisit how requirements raised in the offline (but also online) scenario are addressed by our MSD access control management:

- *Delegation*: Once the `TVD_deploy` protocol [29] has been carried out, the TVD Proxy locally stores an instance of the TVD policy and a certain set of MSD key management records. Hence, it is allowed to enforce the policy and guarantee the access to the subset of MSDs whose keys are stored in the Local Device Directory LDD.
- *Lazy revocation*: Whenever a key revocation occurs, new data, encrypted with the newly generated key, do not overwrite the previous ones, hence, the old data are still available for offline platforms to which the new key has not been delivered.
- *Authentication and integrity* are provided by the identity based signature scheme. Data written to a mobile storage device is digitally signed with the key assigned to the platform the device is attached to. Unauthorized changes afterwards can easily be detected by verifying the signature.
- *Traceability and recovery*: Employing a versioning file service allows to keep track of all modification made to the data, enabling offline platforms to access to the most recent version they can decrypt. Moreover, whenever a revocation occurs, it is possible to retrieve and delete all changes performed by the revoked platform.

6 Implementation

In this section we briefly describe our prototype implementation of the MSD management in a TVD architecture. In particular, we describe our implementation of the virtual storage management. Figure 4 illustrates our implementation.

Our prototype implementation is based on the Turaya [15] security kernel. Turaya is composed of two layers. The first layer is built upon an L4 microkernel [28], which ensures separation among logical execution environments (compartments) and runs services that feature resource management (memory management, I/O). On top of the L4 microkernel, compartments can be native processes (called L4 tasks) or virtual machines (e.g., L4Linux, which is a para-virtualized Linux). The trusted software layer provides security services including secure storage, compartment management, and trusted channel establishment.

In particular, a *trusted channel* [21, 39, 19, 1] is a secure channel established between two compartments

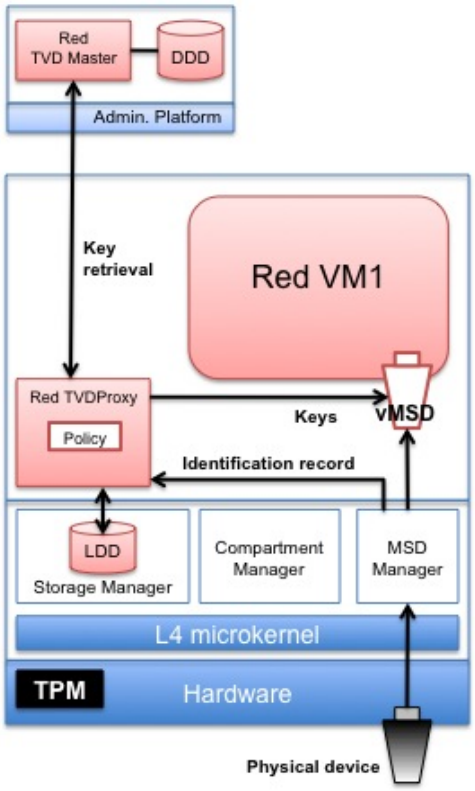


Figure 4: Compartments involved in mounting and accessing to an attached MSD.

that: (a) validate each the configuration and identity of the other, (b) negotiate each an encryption key depending on the configuration of the other, so that data sent over the channel by the former can be accessed only by the latter and *vice versa*. In our architecture, trusted channels are used for the communication between the TVD Master and TVD Proxies. In this way, the TVD Master is ensured that the TVD Proxy to which it is going to send security-critical data, is trusted, and that only that particular TVD Proxy can access data. On the other hand, the TVD Proxy is guaranteed that it is receiving data from the legitimate TVD Master. The *storage manager* is a compartment that provides a persistent storage for other compartments. The storage manager ensures confidentiality, integrity, and freshness of stored data and bind them to the configuration of their owner compartment. For binding data and establishing trusted channels we use trusted computing functionality of a TPM [43].

In addition to these services, our architecture introduces two components that allow the user compartments to access files on devices without having to deal with data encryption themselves.

- The MSD Manager is a compartment, implemented as an L4Linux virtual machine, which handles pluggable USB devices. When a USB disk device is attached, the MSD Manager reads the identification record of the device and forwards it to the TVD Proxy for device recognition and, hence, for key retrieval.

- The *Virtual MSD (vMSD)* maps the physical encrypted device to a virtual (clear) one which is made available to each user compartment of the corresponding TVD. Once the TVD Proxy has retrieved the device keys, it requests the Compartment Manager to create the vMSD, which is implemented as a native L4 task for each MSD. When the device is detached, the vMSD is deleted.

Turaya components plus TVD Master, TVD Proxy, MSD Manager and vMSD compose the Trusted Computing Base of our architecture.

As already exposed in Section 5.4, our architecture stores data on mobile devices through a secure versioning filesystem in which file content is encrypted and each file version is signed by the user that created it. In our early experiments, the versioning filesystem has been implemented on Linux as a filesystem layer based on the FUSE [40] module, making the system independent from the underlying device filesystem format.

The diagram in Figure 6 summarizes the performance measurement performed with the Bonnie benchmark tool [8]. The first column of each group of measurements (version only) shows the respective performance achieved by the filesystem without performing neither encryption nor signature of data. The values summarized by the second column are obtained by running the test enabling cryptographic features on single versioned files. The purpose of this test is evaluating the overhead introduced by the encryption scheme. In the third column we can observe the performance measurement on files that have eight previous versions. In this case, we aimed to point out the overhead due to the signature verification of the previous file versions.

We preliminarily point out that in general performance of devices like USB sticks, memory cards and so on are not very good if compared with the traditional hard disks and that the use of digital signatures (usually avoided in handling contents in secure filesystems) heavily affects performance. On the other hand, we notice that actually mobile devices are mostly used as temporary storage. That is, users prefer to copy documents lying on the pen drive to the local hard drive, edit them there, and then copy them back to the mobile device. Hence, in this usage scenario filesystem performance are not a critical issue. Nevertheless, our tests allowed us to focus on the filesystem reliability, though we find that these results are rather promising and some improvements could be achieved by enhancing the file signature mechanism.

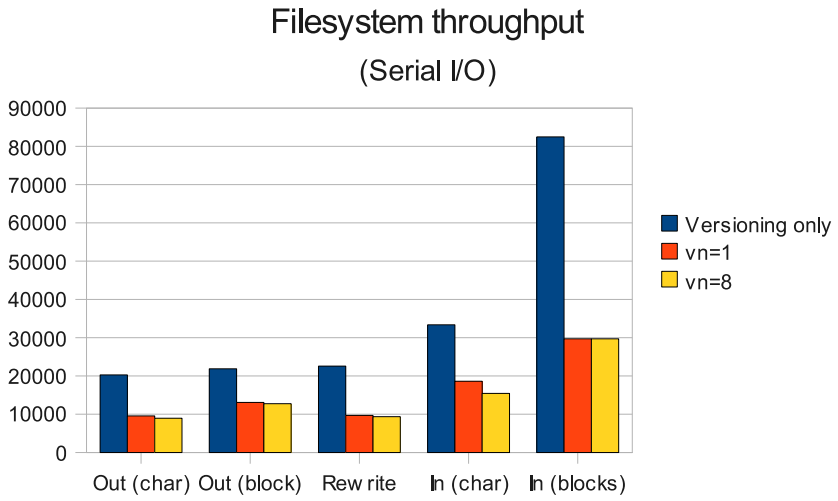


Figure 5: Performance measurement summary. The first column (versioning only) shows the throughput of the filesystem without cryptographic features. The remaining columns show test results on encrypted and signed files, where in average *Vn* versions are present.

7 Security Considerations

In this section we briefly focus on the main security aspects of our architecture, focusing on possible attacks an adversary could launch to untrusted components of our model: user application, devices and offline platforms.

We denote as "adversary" both an outsider entity who is unaware of any information about the network and its TVDs (e.g., old data and keys, namespace conventions etc.), and revoked members with insider information who are no longer trusted and are assumed to act as adversaries.

We assume that adversaries are not able to attack the platform hardware. For example, adversaries cannot examine the content of the memory and possibly extract any information by rebooting the platform and analyzing its memory [24].

Moreover, our scheme aims at preventing disclosure of sensitive data through the misuse of mobile storage devices, whereas it is possible, for example, that insider attackers could make an illegal copy of a confidential document by taking a picture of the screen with a camera, or simply printing it. However, in this work, we do not cope with this kind of threats.

- *Attacks to user applications*. An adversary who exploits a user application running in any user compartment (or the compartment's OS itself), enjoys his victim's access privileges to the plugged MSD. The adversary can obtain and modify data through the application environment. However, the attacker can neither obtain the MSD keys he is using nor can he force the platform to change the keys currently used, as keys and encryption algorithms are not present in the application environment. This threat could be mitigated by letting the TVD Proxy periodically verify the integrity of the user compartment (e.g., each time check if records in the *LDD* are up-to-date). If the verification fails, the TVD Proxy asks the Compartment Manager to kill the hijacked compartment and requests the TVD Master for revocation of the key of the MSD it was using.

- *Attacks to the MSD*. The MSD is a passive device that cannot enforce any effective security measure. An adversary who physically accesses the device, can copy, corrupt and delete both files and the identification record, making the device content no longer available to legitimate users. However, adversaries cannot forge any valid data signature since they have not any valid signing key. Moreover, adversaries cannot read the content of the files stored on the MSD, because they do not possess the necessary encryption keys. However, revoked users can still read data that is encrypted with a key they obtained before being revoked. Roll-back attacks are always possible. Adversaries could make a copy of the filesystem at a certain time and afterwards they can overwrite each more recent version of any file. Revoked users can roll-back the MSD content to a date it was still legitimate and begin an unauthorized branch of the data.

- *Attacks to offline platforms*. Data modifications carrying signatures by revoked users are no longer con-

sidered valid. Offline platforms may still successfully verify signatures by certain revoked users, and even produce some output based on possibly malicious input. However, when the MSD containing such "corrupted data" is connected to an online platform, unauthorized modifications can be found and discarded. Nevertheless, in this way, revoked members might still obtain new data from offline platforms to which the new key has not been delivered yet. Currently, our implementation does not include a solution to this problem.

8 Related Work

The widespread use of Mobile Storage Devices (e.g., memory cards, USB sticks, transportable solid-state hard disks), that allow users to move files among different workstations, poses several problems, *in primis* related to data confidentiality and integrity. In order to cope with these problems, cryptographic mechanisms, i.e., encryption and digital signatures are useful means.

Cryptographic filesystems [7, 12, 27, 25] embed encryption mechanisms into the filesystem operation, featuring a way to encrypt data and metadata without any effort by user level applications. This makes it possible to have good performance and fine-grained security. In particular, the Plutus filesystem [25] features lazy re-encryption [2] at the level of single file-blocks and the *key rotation* mechanism to efficiently generate and manage new encryption keys.

Traditionally, cryptographic filesystems provide a client-server architecture in which the former is trusted and features file content encryption, integrity verification and key management and the latter (untrusted) simply acts as storage for encrypted files. Although several encrypted filesystem can be used also to encrypt local storage devices, they best fit the networked scenario.

Solutions that focus on local storage encryption vary between full disk encryption enforced by hardware or software security modules and creating encrypted partition on local devices [30, 42]. In this case, the aim is guaranteeing the data confidentiality even if the device is stolen and connected to another computer.

The Virtual Private File System [44] leverages on virtualization to assure confidentiality whereas data is accessed through a possibly compromised operating system. Sensitive applications run in a trusted compartment and access their own separated storage through a filesystem layer that features data secrecy, integrity and recoverability and relies on the untrusted filesystem provided by a virtualized legacy operating system.

Encrypted filesystems as those mentioned above are built on top of a specific operating system and are generally not portable. This may introduce inacceptable constrains in a large scale environment. Moreover, distributed encrypted filesystems have, in many cases, their own key management infrastructure which may not be easily interoperable with other existing infrastructures (e.g., PKIs, LDAP). This introduces some redundancies and administrative overhead. Conversely, local storage encryption facilities essentially protect personal devices and workstation and do not feature any distributed key management service. The VPFS also suffers from this shortcoming. In contrast, our solution works for a wide range of applications and operating systems due to the virtualization approach. In fact, any application that can run in a VM transparently benefits from the underlying encryption mechanism. Moreover, it is possible to use the same mobile storage device with its encrypted data on various heterogeneous platforms since the TVD infrastructure provides an abstraction of the underlying encryption mechanism and its key management.

Several architectures aim at enforcing sophisticated security policies within large scale and multi-domain environments and are built on top of a filesystem encryption layer. In particular, the Concord framework [37] allows organizations to monitor data while it is accessed by mobile equipment and makes it possible to enforce the access policies even in a *disconnected* scenario. Institution's data are stored in encrypted form and encryption keys are shared (through a threshold encryption scheme) by a trusted policy enforcer and the user mobile device (e.g., a laptop). In order to access data, the user and the enforcer have to cooperate in order to reconstruct the data encryption key. This approach allows the infrastructure to promptly deny the access to data if it realizes the client has been compromised. In the disconnected scenario, the infrastructure restricts the user privileges to read-only accesses to a subset of organizational data. The role of the enforcer is played by a "disconnected" policy enforcer to which only a limited subset of encryption key shares has been delivered. To the best of our knowledge, Concord is the approach closest to our proposal. In our architecture, Concord's user machine and policy enforcer are collapsed into the same platform, though as different compartments, namely the virtual machine and the TVD Proxy. However, our solution features a less restricted off-line scenario (Concord's disconnected mode does not allow users to modify protected data). Moreover, the virtual storage management in TVDs is in general more flexibile and transparent to the user.

Traceability and reversibility of data modification is an important feature when allowing full data access within the offline scenario and can be achieved through so-called file versioning services, available both at application level [41, 6] and at filesystem level [13, 31, 38, 34]. In particular, several recent proposals address security and integrity checks for stored data, as well as verifiable

audit trails [36, 11, 35]. However, these systems do not fit our requirements since they have not been designed to handle totally passive storage devices.

9 Conclusion and Future Work

In this paper we presented an architecture for the secure and transparent deployment of Mobile Storage Devices (MSDs) within Trusted Virtual Domains (TVDs). We believe that multi-domain IT infrastructures addressed in the TVD model take advantage from the use of these devices. We argued that the usually adopted approaches to protect data stored on these devices suffer from several shortcomings, mainly due to: their intrinsic untraceability, their lack of any effective security feature, and the considerable overhead that their management introduce into the users' work. Moreover, we pointed out that introducing MSDs within TVDs is not a trivial task if the resulting architecture should still be compliant with some typical usage scenarios of these devices, and in particular in the offline scenario. We introduced a general model of a multi-domain environment that enables the secure and transparent use of MSDs and showed how to extend existing TVD architectures with MSD management components to realize this model. Finally, we sketched a proof of concept implementation based on the Turaya security kernel, which uses an L4 microkernel to provide protected execution environments for management services and virtual machines for reusing user applications.

Another important security problem related to MSDs is the proliferation of malicious software like trojan horses and viruses. Our solution limits such attacks, because only data written and signed by legitimate members of a TVD is accepted as input by other TVD members. However, legitimate members might still spread malicious code inadvertently (e.g., if they are infected by a virus). Future research might be directed towards preventing the execution and propagation of malicious code from MSDs.

References

[1] ARMKNECHT, F., GASMI, Y., SADEGHI, A.-R., STEWIN, P., UNGER, M., RAMUNNO, G., AND VERNIZZI, D. An efficient implementation of trusted channels based on OpenSSL. In *Proceedings of the 3rd ACM Workshop on Scalable Trusted Computing (STC 2008)* (2008), ACM Press, pp. 41–50.

[2] BACKES, M., CACHIN, C., AND OPREA, A. Lazy revocation in cryptographic file systems. In *3rd International IEEE Security in Storage Workshop (SISW 2005), December 13, 2005, San Francisco, California, USA* (2005), pp. 1–11.

[3] BACKES, M., CACHIN, C., AND OPREA, A. Secure key-updating for lazy revocation. In *Computer Security - ESORICS 2006, 11th European Symposium on Research in Computer Security, Hamburg, Germany, September 18-20, 2006, Proceedings* (2006), vol. 4189 of *Lecture Notes in Computer Science*, Springer, pp. 327–346.

[4] BEAUTEMENT, A., COLES, R., J., IOANNIDIS, C., MONAHAN, B., PYM, D., SASSE, A., AND WONHAM, M. Modeling the human and technological costs and benefits of USB memory stick security. In *Workshop on the Economics of Information Security (WISE)* (2008).

[5] BERGER, S., CÁCERES, R., PENDARAKIS, D. E., SAILER, R., VALDEZ, E., PEREZ, R., SCHILDHAUER, W., AND SRINIVASAN, D. TVDc: Managing security in the trusted virtual datacenter. *Operating Systems Review 42*, 1 (2008), 40–47.

[6] BERLINER, AND POLK. Concurrent versions system (cvs), 2001. `http://www.cvshome.org/`.

[7] BLAZE, M. A cryptographic file system for unix. In *ACM Conference on Computer and Communications Security* (1993), pp. 9–16.

[8] BRAY, T. Bonnie - filesystem benchmark tool. `http://www.textuality.com/bonnie`.

[9] BUSSANI, A., GRIFFIN, J. L., JANSEN, B., JULISCH, K., KARJOTH, G., MARUYAMA, H., NAKAMURA, M., PEREZ, R., SCHUNTER, M., TANNER, A., DOORN, L. V., HERREWEGHEN, E. A. V., WAIDNER, M., AND YOSHIHAMA, S. Trusted Virtual Domains: Secure foundations for business and IT services. Tech. Rep. RC23792, IBM Research, 2005.

[10] CABUK, S., DALTON, C. I., RAMASAMY, H. V., AND SCHUNTER, M. Towards automated provisioning of secure virtualized networks. In *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, October 28-31, 2007* (2007), ACM, pp. 235–245.

[11] CACHIN, C., AND GEISLER, M. Integrity Protection for Revision Control. In *Applied Cryptography and Network Security: 7th International Conference, ACNS 2009, Paris-Rocquencourt, France, June 2-5, 2009, Proceedings* (2009), Springer, p. 382.

[12] CATTANEO, G., CATUOGNO, L., SORBO, A. D., AND PERSIANO, P. The design and implementation of a transparent cryptographic file system for unix. In *Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference, June 25-30, 2001, Boston, Massachusetts, USA* (2001), USENIX, pp. 199–212.

[13] CORNELL, B., DINDA, P., AND BUSTAMANTE, F. Wayback: A user-level versioning file system for linux. In *Proceedings of Usenix Annual Technical Conference, FREENIX Track* (2004), pp. 19–28.

[14] DISTRIBUTED MANAGEMENT TASK FORCE. "web-based enterprise management (wbem)". `http://www.dmtf.org`.

[15] EUROPEAN MULTILATERALLY SECURE COMPUTING BASE(EMSCB) PROJECT. Towards trustworthy systems with open standards and trusted computing, 2008. `http://www.emscb.de/`.

[16] EUROPEAN NETWORK AND INFORMATION SECURITY AGENCY (ENISA). Secure USB Flash Drives, June 2008. `http://www.enisa.europa.eu/doc/pdf/publications/SecureUSBdrives_180608.pdf`.

[17] FABIAN, M. Endpoint security: managing USB-based removable devices with the advent of portable applications. In *InfoSecCD '07: Proceedings of the 4th annual conference on Information security curriculum development* (New York, NY, USA, 2007), ACM, pp. 1–5.

[18] GALINDO, D., HERRANZ, J., AND KILTZ, E. On the generic construction of identity-based signatures with additional properties. In *Advances in Cryptology - ASIACRYPT 2006, 12th International Conference on the Theory and Application of Cryptology and Information Security, Shanghai, China, December 3-7, 2006, Proceedings* (2006), vol. 4284 of *Lecture Notes in Computer Science*, Springer, pp. 178–193.

[19] GASMI, Y., SADEGHI, A.-R., STEWIN, P., UNGER, M., AND ASOKAN, N. Beyond secure channels. In *Proceedings of the 1st ACM Workshop on Scalable Trusted Computing (STC'07)* (2007), ACM Press, pp. 30–40.

[20] GASMI, Y., SADEGHI, A.-R., STEWIN, P., UNGER, M., WINANDY, M., HUSSEIKI, R., AND STÜBLE, C. Flexible and secure enterprise rights management based on trusted virtual domains. In *Proceedings of the 3rd ACM Workshop on Scalable Trusted Computing, STC 2008, Alexandria, VA, USA, October 31, 2008* (2008), ACM, pp. 71–80.

[21] GOLDMAN, K., PEREZ, R., AND SAILER, R. Linking remote attestation to secure tunnel endpoints. In *Proceedings of the First ACM Workshop on Scalable Trusted Computing (STC'06)* (2006), pp. 21–24.

[22] GRIFFIN, J. L., JAEGER, T., PEREZ, R., SAILER, R., VAN DOORN, L., AND CÁCERES, R. Trusted Virtual Domains: Toward secure distributed services. In *Proceedings of the 1st IEEE Workshop on Hot Topics in System Dependability (HotDep'05)* (June 2005).

[23] GUILLOU, L. C., AND QUISQUATER, J.-J. A "paradoxical" indentity-based signature scheme resulting from zero-knowledge. In *Advances in Cryptology - CRYPTO '88, 8th Annual International Cryptology Conference, Santa Barbara, California, USA, August 21-25, 1988, Proceedings* (1990), vol. 403 of *Lecture Notes in Computer Science*, Springer, pp. 216–231.

[24] HALDERMAN, J. A., SCHOEN, S. D., HENINGER, N., CLARKSON, W., PAUL, W., CALANDRINO, J. A., FELDMAN, A. J., APPELBAUM, J., AND FELTEN, E. W. Lest we remember: cold-boot attacks on encryption keys. *Commun. ACM 52*, 5 (2009), 91–98.

[25] KALLAHALLA, M., RIEDEL, E., SWAMINATHAN, R., WANG, Q., AND FU, K. Plutus: Scalable secure file sharing on untrusted storage. In *Proceedings of the FAST '03 Conference on File and Storage Technologies, March 31 - April 2, 2003, Cathedral Hill Hotel, San Francisco, California, USA* (2003), USENIX.

[26] KATSUNO, Y., KUDO, M., PEREZ, P., AND SAILER, R. Towards Multi-Layer Trusted Virtual Domains. In *The 2nd Workshop on Advances in Trusted Computing (WATC 2006 Fall)* (Tokyo, Japan, Nov. 2006), Japanese Ministry of Economy, Trade and Industry (METI).

[27] LI, J., KROHN, M. N., MAZIÈRES, D., AND SHASHA, D. Secure untrusted data repository (sundr). In *OSDI* (2004), pp. 121–136.

[28] LIEDTKE, J. On micro-kernel construction. In *SOSP* (1995), pp. 237–250.

[29] LÖHR, H., SADEGHI, A.-R., VISHIK, C., AND WINANDY, M. Trusted privacy domains – challenges for trusted computing in privacy-protecting information sharing. In *Information Security Practice and Experience, 5th International Conference, ISPEC 2009* (2009), vol. 5451 of *Lecture Notes in Computer Science*, Springer, pp. 396–407.

[30] MICROSFOT CORP. Bitlocker drive encryption, 2006. `http://technet.microsoft.com/en-us/windows/aa905065.aspx`.

[31] MUNISWAMY-REDDY, K., WRIGHT, C. P., HIMMER, A., AND ZADOK, E. A Versatile and User-Oriented Versioning File System. In *Proceedings of the Third USENIX Conference on File and Storage Technologies (FAST 2004)* (San Francisco, CA, March/April 2004), USENIX Association, pp. 115–128.

[32] NAOR, D., SHENHAV, A., AND WOOL, A. Toward securing untrusted storage without public-key operations. In *Proceedings of the 2005 ACM Workshop On Storage Security And Survivability, StorageSS 2005, Fairfax, VA, USA, November 11, 2005* (2005), ACM, pp. 51–56.

[33] PARKIN, S. E., KASSAB, R. Y., AND VAN MOORSEL, A. P. A. The impact of unavailability on the effectiveness of enterprise information security technologies. In *Service Availability, 5th International Service Availability Symposium, ISAS 2008, Tokyo, Japan, May 19-21, 2008, Proceedings* (2008), vol. 5017 of *Lecture Notes in Computer Science*, Springer, pp. 43–58.

[34] PETERSON, Z., AND BURNS, R. Ext3cow: a time-shifting file system for regulatory compliance. *ACM Transactions on Storage (TOS) 1*, 2 (2005), 190–212.

[35] PETERSON, Z., BURNS, R., ATENIESE, G., AND BONO, S. Design and implementation of verifiable audit trails for a versioning file system. In *Proceedings of the 5th USENIX conference on File and Storage Technologies table of contents* (2007), USENIX Association Berkeley, CA, USA, pp. 20–20.

[36] SHAPIRO, J. S., AND VANDERBURGH, J. Access and integrity control in a public-access, high-assurance configuration management system. In *Proceedings of the 11th USENIX Security Symposium, San Francisco, CA, USA, August 5-9, 2002* (2002), USENIX, pp. 109–120.

[37] SINGARAJU, G., AND KANG, B. H. Concord: A secure mobile data authorization framework for regulatory compliance. In *Proceedings of the 22nd Large Installation System Administration Conference, LISA 2008, November 9-14, 2008, San Diego, CA, USA* (2008), USENIX Association, pp. 91–102.

[38] SOULES, C., GOODSON, G., STRUNK, J., AND GANGER, G. Metadata Efficiency in Versioning File Systems. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies* (2003), USENIX Association Berkeley, CA, USA, pp. 43–58.

[39] STUMPF, F., TAFRESCHI, O., RÖDER, P., AND ECKERT, C. A robust integrity reporting protocol for remote attestation. In *2nd Workshop on Advances in Trusted Computing (WATC'06 Fall)* (Tokyo, December 2006).

[40] SZEREDI, M. File system in user space. `http://sourceforge.net/projects/fuse`.

[41] TICHY, W. Design, implementation, and evaluation of a Revision Control System. In *Proceedings of the 6th international conference on Software engineering* (1982), IEEE Computer Society Press Los Alamitos, CA, USA, pp. 58–67.

[42] TRUECRYPT FOUNDATION. Truecrypt - free open-source on-the-fly encryption, 2004. `http://www.truecrypt.org/`.

[43] TRUSTED COMPUTING GROUP. TPM main specification, version 1.2 rev. 103, July 2007. `https://www.trustedcomputinggroup.org`.

[44] WEINHOLD, C., AND HÄRTIG, H. VPFS: Building a virtual private file system with a small trusted computing base. In *Proceedings of the 2008 EuroSys Conference, Glasgow, Scotland, UK, April 1-4, 2008* (2008), ACM, pp. 81–93.

[45] WIRED. Under worm assault, military bans disks, USB drives, Nov. 2008. `http://www.wired.com/dangerroom/2008/11/army-bans-usb-d`.